UNIVERSITY OF OSLO
Department of Informatics

Video Quality Measurement of Scalable Video Streams

Master thesis

Bjørn Olav Ruud

02.05.2006

Abstract

Video streams in general do not adapt to changes in the network environment, causing image quality to suffer. Graceful video scaling requires fine granular adaption, and scalable video codecs like SPEG and MPEG-4 FGS provide this ability. Little research has been done regarding how continuous quality changes affect perceived video quality, and how existing metrics can be used to measure this. There is no objective metric that addresses this particular problem.

Some objective metrics emulate the human visual system, and the project compared those with subjective results. The comparison indicated objective tests could be used in place of subjective tests, which are more resource expensive. More testing was needed to draw any conclusions about this topic.

Results from the quality evaluation tests could be used to generate a utility function for measuring perceived quality in scalable video. It would be based on exponential functions describing each test parameter. Due to time constraints, only an abstract method to approach this problem was proposed.

Some shortcomings were observed. The test parameters were not numerous enough, and their values were not distanced far enough apart. Testing performed with the chosen subjective evaluation method, DSCQS, resulted in a data set that was too small for serious usage. The test should have had access to more participants, and the viewers should have seen fewer clips but with more parameter variations.

Foreword

Acknowledgments go to my student advisors: Svetlana Boudko (Norwegian Computing Center) and Carsten Griwodz (IFI, University of Oslo).

Technical Notes

The thesis was written in LaTeX using the book style with 11pt Times font and A4 paper size. Source code in the appendix was formatted with the *listings* package.

Contents

Chapter 1

Introduction

A digital video stream will often be transferred over networks where conditions vary greatly and are not directly controllable by the sender. In addition the stream will likely be displayed on a large variety of devices, each with a different set of resources. Thus arises the need for adjusting or degrading the stream in response to varying network conditions, device capabilities or some other measure, so that a perceived quality is attained that is within an acceptable range for the service provided. This means a quality trade-off must be made, either as a preventive measure or as a real-time response to changes in the transmission environment.

There are many metrics for measuring video quality, both objectively using machine evaluation and subjectively using human evaluation. Some objective methods approximate how the human visual system works in order to get similar results as subjective methods. All of these metrics were created to evaluate video with static parameters, e.g. the framerate does not change for the duration of the video clip. Measuring scalable video, which has continuous parameter adjustments, requires a new metric. This thesis will propose a way to approach this problem by using existing metrics.

Another topic will be how suitable objective metrics are as substitutes for subjective testing. This is useful because of resource issues, since subjective testing are difficult to organize and are time consuming. The project will try to perform research and tests to make this comparison possible. Not all objective metrics are suitable for this, and the thesis will concentrate on those that try to emulate the human visual system.

The thesis is aimed at new Master students with a basic background in video compression and networking. In chapter 2 an overview is given for relevant topics in video compression, computer networks and video quality measurement methods. Chapter 3 details how all tests, objective and subjective, were designed and executed in order to gather data. The results are then analyzed and discussed in chapter 4, before some conclusions and comments are made in chapter 5.

Chapter 2

Background

In order to approach the problem of creating a video quality metric for scalable video, one needs to understand the technologies and techniques used by such video systems, and how to evaluate video in general. Compression, scaling and evaluation can all be done in a number of ways. The compression method will impact what kind of impairments is introduced to the video stream. How the video is scaled will either hide or enhance these visual artifacts, affecting how overall quality is perceived. The video quality can be evaluated both objectively and subjectively depending on the area of application. For the project MPEG-4 video compression was used, and an overview of MPEG compression in general will be described.

2.1 Video Compression

The goal of any compression is to reduce the size of the data as much as possible while retaining the information that is considered necessary. Lossless compression must be able to restore the data to its original state. These methods transform the data to a representation that saves space, for instance by using short-hand codes for recurring patterns. On the other hand, with lossy compression the removal of information to achieve a higher compression ratio is acceptable. The encoding process then becomes a trade-off between size and how representative of the original data the compressed version should be.

JPEG

Central to MPEG video compression is the JPEG [1] image compression method for still images. It works by removing redundant image data in a way that tries to stay pleasing to the human eye. JPEG compression involves several processing steps:

- *Color space conversion.* JPEG compression is not performed in RGB color space, but in YUV, having one component channel for luminance (Y) and

two for chrominance (U and V). This color space was created to more closely model the human perception of color than RGB. Fig. 2.1 and fig. 2.2 show the conversion algorithm using equations and matrices respectively. Because the human eye is less sensitive to high frequency color information compared to luminance, the chrominance components are sampled at half the resolution of the luminance component to increase compression.

- *DCT transform*. Each of the channels from the color space conversion are transformed from the spatial domain to the frequency domain by performing the Discrete Cosine Transform (DCT). The DCT is preferable to a Fourier Transform (FT) as it has a energy compaction property, where most of the signal information end up in a relatively small set of low frequency components. This transformation is applied to 8 by 8 pixel blocks.

- *Quantization*. This step is where most of the actual compression takes place. The resulting values from the DCT are divided with a constant to get a smaller value (which requires less bits to represent), and rounded to the nearest integer. A pre-defined 8x8 quantization matrix contains the constants, meaning there is one for each component of the 8x8 block. Since the human eye is less sensitive to high frequency components those are divided with larger constants. Because of this the resulting 8x8 block will contain many zero values in the high frequency part, which enables better compression in the next stage.

- *Entropy coding*. The result from the quantization step can be further compressed by re-organizing the data. The elements of a 8x8 block is first ordered in an array by moving through the values in a zig-zag manner. This puts the low frequency values at the end, and most of those are likely to be zero. Run-length encoding is then applied. Finally Huffman coding is used, which is a lossless procedure. It works by representing frequently used symbols with short codes, and using longer codes as the symbols become more rare.

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= -0.147R - 0.289G + 0.436B \\ V &= 0.615R - 0.515G - 0.100B \end{aligned}$$

Figure 2.1: RGB to YUV conversion (equation)

MPEG Video

For video it is certainly possible to simply use JPEG compression on a per frame basis, as implemented in MJPEG (Motion-JPEG). For some areas of application

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Figure 2.2: RGB to YUV conversion (matrix)

(like video editing) this is even encouraged, since you get frame accurate edits, easy synchronization and predictable quality and bandwidth management. However, despite managing a decent compression ratio for single images, size can quickly become a problem in a video context, as you are dealing with several frames per second (25 for PAL). The problem lies with not exploiting the temporal nature of video. The changes between frames in a scene are usually small enough that encoding a whole new frame becomes inefficient. This is the problem that MPEG video compression addresses, by introducing methods to not only compress spatially but also temporally. The following describes MPEG-2 [24] video with additions for the video part of MPEG-4 where appropriate.

MPEG video uses three frame types, I-, P- and B-frames. I-frames, or intra frames, function as keyframes, and are simply a JPEG compressed image. P-frames, or predicted frames, are based on the most recent reconstructed I- or P-frame. The differences are recorded as either a motion vector and difference DCT coefficients, or simply an intra coded block if there is no good match. B-frames, or bidirectional frames, use both a forward and a backward reference in time to the closest I- or P-frames. Differences are recorded as motion vectors based on one of the referenced frames, or intra coded by subtracting the average between them from the block being coded. A frame sequence could look like this: `IBBPBBPBBI`.

2.2 Scalable Video

Video streams on the Internet are typically not scalable. Usually the streams are offered at a number of fixed rates, and the user chooses one of these rates before streaming begins. Some implementations offer automatic switching between the streams based on available bandwidth [28]. The fixed rate approach has several negative aspects:

- If there is more bandwidth available it is not utilized, so the stream can't scale upwards in quality.
- If bandwidth drops below the minimum needed the stream can't be downscaled either, so it starts losing data, possibly making the service unavailable.
- Since there are no levels or no hierarchy describing data importance, the streams do not recover gracefully from missing data.

In the case of the scheme using automatic stream switching, available resources will likely limit us to a rather small number of streams. Each of them has to be encoded separately and the more streams you have the more encoding time and storage space is used. A small number of streams will give a coarse granularity resulting in noticeable jumps in quality. This is represented by fig. 2.3, where the smooth curve represents actual needed bandwidth for a given quality level, and the lines represent fixed bitrate streams. As bandwidth increases it is not utilized until there is enough for the next step, and for a fixed rate stream the extra bandwidth isn't used at all. The curve represents the optimal quality for a given bandwidth, and the goal for a scalable video codec is to stay as close to this curve as possible. Take note that the shape of the curve indicates a larger gain in quality by adding bits at low rates, and as the bitrate rises the observed gain in quality decreases until a point where it is the same as the unscaled source material.

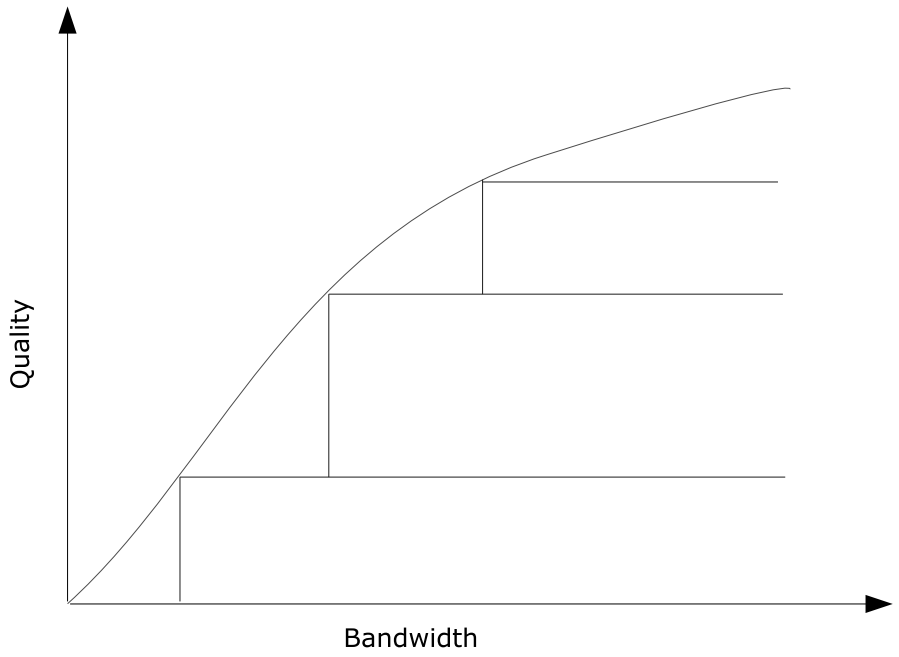

Figure 2.3: Bandwidth vs Quality

Non-scalable video codecs are not suitable for transmission over heterogenous networks like the Internet. For instance, if the combined traffic exceeds capacity for a line the rate of all streams will be reduced, which is commonly done by discarding packets. When packet loss occurs the end-user will experience a delay, video corruption or even complete loss of video for the affected parts of the stream. Some video codecs use frames that reference data found forward or backward in

the stream, and might get more widespread corruption. Also, most video encoders today provide a variable bitrate approach to save space while maintaining high quality, and this causes the video stream to have a bursty behavior when it comes to bandwidth usage. The bandwidth usage of such streams are difficult to predict.

A scalable video codec will have more success under circumstances as described above. The codec will have the video data organized in a layered or hierarchical manner, so packet loss or bandwidth reduction will not necessarily mean that we get a lost or corrupt video frame, but a degraded one. This degradation is ideally done in a graceful manner by using techniques to achieve a highly granular scalability.

There are several scalability techniques available for scalable video codecs, and they can be divided into three main classes:

- *SNR scalability*. Layers have the same frame rate and spatial resolution, but different quantization accuracy. Less precision means more visual artifacts (blocks, gradients, etc.).
- *Temporal scalability*. Layers have the same spatial resolution but different frame rates. Here smoothness of the video is sacrificed to gain a higher quality per frame.
- *Spatial scalability*. Layers have the same frame rate but different spatial resolutions. Less pixels used to describe the image means less detail is preserved.

A scalable video implementation usually implements a combination of these techniques, as the type of source material and its intended use can affect which to choose. For instance, animation is usually drawn at a less than full frame rate, and does not contain as much detail as a photo. On the other hand, the smooth gradients, flat colored surfaces and sharp line contrasts which are numerous in this kind of source material will make quantization errors stand out more than usual. In this example preserving quantization accuracy while decreasing the frame rate and spatial resolution should work well.

2.2.1 SPEG

Scalable MPEG, or SPEG [18], was created to be adaptable through a wide range of rate and quality levels, and for simplicity only SNR scalability has been implemented. Normally a MPEG encoder creates streams aimed at a static target bitrate, and have a rate control mechanism that tries to find quantization values for the blocks in a frame that results in a match for this rate (averaged over a small interval). The chosen bitrate is an estimation of a safe rate which there will be available bandwidth for, in order to avoid image corruption. If a worst case scenario is used as a basis, the bitrate will be lower than what the network can support, and bandwidth will be wasted. A spatially layered coding like SPEG can instead target a

higher bitrate by default by partitioning the coefficient data so that quantization can be done dynamically and incrementally. This means that we can dynamically adjust the bitrate based on some feedback mechanism, and to do this adjustment we essentially change the precision of the coefficient values by dropping data of lesser importance according to some hierarchy or ranking, resulting in loss of spatial fidelity similar to when using higher quantizers. The end result is a reduction in image quality but we avoid data loss or visual corruption which would have happened in a static bitrate scheme.

The MPEG coefficients are transcoded to a base level and a number of enhancement levels. In default SPEG there are three enhancement levels, but the method scales upwards to the number of bits used for the values minus one, if desired. The data is partitioned in a bit-plane like manner, where each level only contains the bits from the corresponding bit position in the stream. Bit-planes are described in the next section. It is important to note that SPEG uses a hierarchical layer scheme, so lower levels must be present for the ones higher up to be used. Hence if data from a lower level is damaged or missing all data that depends on it will be discarded. A nice property of the SPEG encoding method is that it can be reversed back to the original MPEG stream.

Other scalability options for future use are spatial size and chroma scalability. A weakness in MPEG (and hence SPEG) rate control is that we do not know how much image degradation a certain level of quantization causes. For details about SPEG, refer to Krasic's original paper [18].

2.2.2 MPEG-4 FGS

SPEG is sufficient for demonstration purposes, but is not as efficient as MPEG-4 FGS [21]. Since SPEG was meant as a reference codec to test the capabilities of the QStream framework, it was not optimized for high compression efficiency and other methods of scaling beside SNR. MPEG-4 FGS was designed to go further than SPEG in both efficiency and scalability options.

The FGS reference specification uses a layered data approach similar to SPEG by using one base layer and one enhancement layer. The layers are based on the same bit-plane data partitioning principle as SPEG but offers spatial and temporal scalability in addition to SNR scalability. These options have been described earlier, but there are some FGS specific details:

- SNR scalability is achieved by having the same framerate and spatial resolution in the layers, but different quantization accuracy. The enhancement layer contains the DCT coefficient difference from the base layer, and the values are simply added. The coding efficiency will depend on the use of the enhancement layer in the encoder and decoder.

- Temporal scalability uses layers with the same spatial resolution but different frame rates. The base layer has a lower frame rate than the enhancement layer, and it is the enhancement layer which provides the missing frames

to achieve full frame rate. Only P-type prediction is used in the base layer, while the enhancement layer can use P- or B-type referenced from the base layer, or P-type references from the enhancement layer.

- Spatial scalability has layers at the same frame rate, but different spatial resolutions. The base layer will in this case have lower resolution than the enhancement layer, and will be upsampled to the same resolution as the enhancement layer before being used.

In SPEG an enhancement layer is either used or not used at all. FGS improves upon this by enabling only a part of the enhancement layer to be used. The data in the layer can be truncated to gain the ability to do continuous scalability.

The primary component FGS uses to achieve fine granular scalability is bit-plane coding, and the key to bit-plane coding is to see the DCT coefficients as a binary number of several bits instead of a decimal integer. Each 8x8 DCT block is zig-zag ordered into an array. Arrays of 64 bits are created to represent a bit-plane (MSB, MSB-1, ...) and filled with the appropriate bits from the DCT values.

As an example, consider the following array of decimal values and their corresponding sign bits:

```
4,0,1,2,3 (absolute)
0,0,1,0,0 (sign)
```

The values are then converted to a binary sequence, with the bits placed vertically, most significant bit (MSB) first:

```
1,0,0,0,0 (MSB)
0,0,0,1,1 (MSB-1)
0,0,1,0,1 (MSB-2)
```

Each row in this matrix is a bit plane, and contains all the bit values of a certain significance. In other words, the bits are grouped by precision. Removing the least significant bit (LSB) will not change the value much but will lower accuracy. In order to efficiently compress these bit-planes, they are converted to (RUN, EOP) pairs, where RUN is number of consecutive zeros before a 1 and EOP is whether there are any more 1 values on the plane (with 1 meaning there are no more). With traditional run-length encoding (RLE) there is some form of (COUNT,SYMBOL) notation, i.e. a run of 15 A letters will be (15,A). Compared with the FGS method this will result in more entries and increase final data size. By example, the RLE version of the above bitplanes would look like this:

```
(1,1),(4,0)
(3,0),(2,1)
(2,0),(1,1),(1,0),(1,1)
```

Very short runs are penalized with RLE but not so much with the FGS method:

```
(0,1)
(3,0),(0,1)
(2,0),(1,1)
```

There are also some variations on FGS video encoding possible within the standard, as mentioned in [21]:

- *Different Numbers of Bit-Planes for Individual Color Components*. The YUV color components may have different number of bit-planes, so the maximum for each are encoded in the frame header.

- *Variable-Length Codes*. The MSB plane is defined on a block-by-block basis as the first plane that is not all-zero. All-zero planes can be coded efficiently by grouping the blocks in each macroblock and code the all-zero cases together.

- *Decoding Truncated Bitstreams*. The input stream to a FGS decoder can be truncated, and decoding of such a stream is not standardized in MPEG-4. One can look ahead 32 bits at every byte aligned position in the stream, and synchronize on `fgs_vop_start_code`.

The MPEG-4 video standard is complex and contains many compression methods. It was never the intention that all of them must be implemented or used, so a set a profile definitions was defined. Both encoders and decoders can then reduce complexity by only implementing the methods that belong to the profiles they intend to support. For FGS, the base layer must adhere to the Advanced Simple profile, which is the same as defined for non-scalable video. The enhancement layer has its own profile definition, simply called FGS. In addition to the standard encoding methods FGS has some advanced (but optional) methods to increase visual quality:

- *Frequency weighting*. Visually the accuracy of the low frequency DCT coefficients are usually more important than the high. The low frequency DCT components can be put earlier into the bitstream so they are more likely to be included should the bitstream be truncated. Additional VLC tables are needed for coding.

- *Selective enhancement*. Parts of frames may be more important than others. The macroblocks in question can be bit-plane shifted so they are more likely to be included in a truncated bitstream.

- *Error resilience*. Wireless environments might experience random burst errors in the bitstream. Resynchronization markers in the enhancement layer can be used to quickly isolate the error.

- *Temporal scalability*. To cover a wide bitrate range there is a need for scalable temporal resolution [27, 32] (frame rate). The prediction of FGST frames are calculated only from the base layer, and also have scalable quantization accuracy within each temporal enhancement frame. FGST frames can be a part of the enhancement layer or stored in a separate layer.

To summarize, MPEG-4 FGS is a result of the growing need for delivering video over a wide range of bit rates and bit rate variations. Its goals are high coding efficiency and low implementation complexity. Bit-plane coding is efficient and able to deliver graceful changes in quality as the bitrate changes. FGS separates encoding from transmission as the enhancement layer can be adapted for transmission at any bitrate without transcoding.

2.3 Network

The behavior of a video codec under different network conditions is generally known, and if not can usually be derived, but the effect on perceived quality can not directly be interpreted from this behavior. Changes to the network stream result in an altered video presentation for the viewer, and so it is necessary to know how these changes affect perceived quality. What is needed, based on the codec properties, is a mapping between network metrics and quality such that perceived quality can be measured by using network metrics alone. Video streaming providers will most likely have access to statistics from the underlying network and/or client, and may want to adjust streams to maintain a certain quality of service. From this a need arises to be able to measure and adjust visual quality from the source.

2.3.1 Metrics

Network parameters of importance are typically bandwidth, packet loss, jitter and latency. All of these have an effect on streaming video, but for our tests latency will not be a factor since none of the tests involve time-critical tasks in the sense of quick response to the users actions, such as starting and stopping the video and seeking. Pre-made streams shown without user interaction, as used by the project, will not be affected by latency, and will not have an impact on visual quality. How these parameters can have an effect on a video stream needs to be defined:

- *Bandwidth*. Its importance is obvious; a change in bandwidth equals a change in the amount of data we can send, and the amount of data dictates how much detail and precision can be preserved. As seen in fig. 2.3 the video quality does not increase linearly together with bandwidth, as it flattens out when we approach the same level as the source material.

- *Packet loss*. Depending on the protocol used this property will have various effects. Many streaming video applications assume delivery on time is more

important than guaranteed data arrival, and use UDP. If a UDP packet gets lost then data will be missing, causing visual corruption, missing parts of the image, or in the worst case whole frames must be discarded. For a scalable codec the damage can be minimized by using less precise values for the decoded image. The visual quality will be lower but at least the frame can be shown. On the other hand, TCP tries to guarantee packet delivery until a connection is considered broken. Packet loss will trigger the sender to try sending the missing packets again, causing data delivery to be delayed. This variation in packet arrival time is seen as jitter. Given a high enough packet delay the decoder will have to pause until enough data has been received to continue, essentially interrupting the flow of video presentation.

- *Jitter*. This occurs when bandwidth and/or packet loss fluctuates a lot. Video streams are time sensitive, as there is only a certain amount of time available to receive data for a frame and display it. Incomplete data might be discarded, and data which is too late isn't used at all. This network parameter can result in frequent quality changes, which will likely degrade perceived image quality. It can also affect the streaming rate controller, causing it to oscillate when trying to average the bitrate would have been preferable.

2.3.2 Network Classes

The simulated network environments the tests in this thesis will be based on should represent typical networks where video streaming is used today, as well as what might be the case in the near future should scalable video streaming become more mainstream. There are especially two market segments of particular interest:

- Wireless networks of the very low bandwidth variety, as used with mobile devices. Typical speeds are what you get with UMTS and EDGE mobile phone technologies. Common problems are a relatively high degree of packet loss, and as a consequence some amount of jitter. Most of all the problem is the low bandwidth limit, seen from a video streaming perspective, though it is somewhat alleviated by the fact that the display devices used to view the video are small in both size and amount of pixels, and do not offer the best visual quality in general.

- Wired networks of the type commonly used for internet access. These networks do not usually provide LAN speeds, but will be used for video streaming in many modern applications. Beneficial properties are low degree of packet loss, and little inherent jitter. This class of networks often has users displaying the video on large displays with a high pixel count, and which can produce excellent image quality. Today many of these displays are HDTV capable, which poses a real challenge with the bandwidth available.

2.4 Image Quality Measurement

There are several ways of measuring video quality, but the methods can generally be divided into those using human evaluation and those that do not. Human, or subjective, evaluation is excellent when a full model of the human visual system is needed. The drawbacks are statistical accuracy and resource requirements. No two humans perceive an image in the exact same way, so a large number of people need to be tested to find a general measure for how that image is perceived. Time and availability of people then become resource issues.

Non-human evaluation use algorithms to measure image error in some way, modeled after the human visual system (HVS) or not. Since no people are needed and the processing can be automated, time is the only resource limitation. However, these test methods do not necessarily model the HVS to its full extent, if they do so at all. The results might not be representative of human visual perception.

The objective and subjective video quality measurement methods will only be explained in brief, as some of them are complex and beyond the scope of this thesis.

2.4.1 Objective Methods

Objective measurements compare the impaired stream with a reference stream using some algorithm. This is fine if the difference between streams is to be evaluated on the basis of some criteria like noise or structural similarity, but does not truly describe how these differences are perceived subjectively. Some of these methods are HVS based and are better at indicating what a subjective observation would be like. Although, because of its complexity, these methods do not incorporate a complete model of the HVS.

Mean Squared Error (MSE)

The MSE is one of the simplest methods to use. One calculates the error of the chosen sample values, square them so negative values do not cancel positive ones, and divide by number of samples. The result indicates the degree of error in the stream when compared to some reference, but says nothing of how this affects perceived quality. The MSE for two mxn images I and K can be expressed like this:

$$MSE = \frac{1}{mn} \sum_{i}^{m} \sum_{j}^{n} \|I(i,j) - K(i,j)\|^2$$

Peak Signal-to-Noise Ratio (PSNR)

PSNR is a term for the ratio between the maximum value of a signal and the magnitude of background noise. A higher ratio yields a cleaner signal. So in video

terms, a high ratio will offer an image with little noise which is closer to the original reference. The calculation of PSNR is based on MSE (MAX is maximum pixel value):

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

Structural Similarity Index (SSIM)

The main problem with MSE and PSNR is that they are based solely on error sensitivity. An image can have very different errors that will be perceived differently, but still have the same metric score. SSIM [37] is a relatively new HVS based objective metric which measures error as the degree of structure distortion. It is based on the assumption that the HVS is adapted to identify structure in an image. Structure is here defined as pixels with strong dependencies, especially spatially proximate ones. An example is a black line on white background. The contrasting values of the pixels along the line edge are what enables us to perceive the structure. Change them and the structure becomes less apparent, and even erased, so they are clearly dependent on each other to define structure.

SSIM uses a combination of luminance, contrast and structure measurement to calculate a quality score. The following is a general overview of the algorithm. First, luminance is estimated as the mean intensity:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i$$

The luminance comparison function $l(x, y)$ is then a function of μ_x and μ_y. To find estimated signal contrast, standard deviation is used:

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}}$$

The contrast comparison $c(x, y)$ is then the comparison of σ_x and σ_y. Last, the structure comparison $s(x, y)$ is performed on the normalized signals $(x - \mu_x)/\sigma_x$ and $(y - \mu_y)/\sigma_y$. Finally the three components are combined to yield the overall similarity measure:

$$S(x, y) = f(l(x, y), c(x, y), s(x, y))$$

VQM

The VQM video quality metric [40] rates the difference between two clips by using the discrete cosine transform (DCT), and is based on earlier research in the field [38]. Because it employs a spatial-temporal masking function that takes into

account how the early stages of the human eye works, this metric can also be considered HVS based. The following steps are taken to calculate the VQM score (summarized from [40]):

1. Color transform. The color space from the original video material is converted to YUV.

2. DCT transform. The image data is separated into different spatial frequency components in the same way as in JPEG and MPEG encoding.

3. Each DCT coefficient is converted to local contrast (LC) using the following equation:

$$LC(i,j) = DCT(i,j) * Power(DC/1024, 0.65)/DC$$

   DC is the DC component of each block. For an 8-bit image, 1024 is the mean DCT value. 0.65 is the best parameter for fitting psychophysics data. After this step, most values lie between [-1,1]. The steps taken so far are identical to Watsons DVQ [38] model.

4. LC is converted to just-noticeable differences (jnds) by applying temporal filtering and the human spatial contrast sensitivity function (SCSF).

5. Weighted pooling of mean and maximum distortion. First the difference between the two sequences is found by subtraction. Then contrast masking is incorporated into a simple maximum operation, and the result is weighted with the pooling mean distortion. This reflects the fact that a large distortion in one region will suppress sensitivity to smaller distortions.

$$\begin{aligned} Dist_{mean} &= 1000 * mean(mean(abs(diff))) \\ Dist_{max} &= 1000 * maximum(maximum(abs(diff))) \\ VQM &= Dist_{mean} + 0.005 * Dist_{max} \end{aligned}$$

   The choice of 0.005 as maximum distortion weight is based on several primitive psychophysics experiments. Parameter 1000 is the standardization ratio.

2.4.2 Subjective Methods

Subjective measurements can generally be divided in two categories: double stimulus and single stimulus. In the first type of method the viewer is exposed to both an impaired stream and a reference stream in random order, and evaluates the quality difference or change in quality. The latter method uses only the impaired video stream, and the subject evaluates quality over time. Single stimulus methods are useful when there is no reference available, or the viewer has to watch very long video sequences. An overview of common subjective measurement methods, their properties, and results from some tests using them can be found in [25].

Double Stimulus Continuous Quality Scale (DSCQS)

In this subjective quality measurement metric the subjects are shown clips in pairs, where one is the reference and the other is an impaired version. The clip order is initially randomized, and that order is used for each subsequent viewing of the clip pair. The pairs are shown at least twice before an evaluation is done. To quantify changes in quality one calculates the difference between the results for the two clips.

The score from this test is known to not be significantly impacted by memory-based biases from previously viewed video sequences. It is also widely accepted as having little sensitivity to context effects. Context effects occur when subjective ratings are influenced by the severity and ordering of impairments within the test session. A problem that can occur is when viewers switch the scores. This will not change how the subject rated the difference between the clips, but gives a wrong rating on a per clip basis. This situation is often visible in the statistics and can be accounted for.

Double Stimulus Comparison Scale (DSCS)

This metric uses pairs of video clips in the same manner as DSCQS, but the pair is shown only once. The main difference between the methods lies with how the pairs are rated. In DSCS the subject is not asked to rate each clip in the pair, but instead evaluates the difference between them and translates that to a single score on a seven point discrete scale. This way it doesn't matter if the subject is confused about score ordering, but with only a single viewing the score accuracy might be lower.

Single Stimulus Continuous Quality Evaluation (SSCQE)

Single stimulus methods are based on rating a continuous video clip, with no reference comparison. These methods are needed for situations where the reference is not available, e.g. client side video quality measurement. A common method is SSCQE, in which subjects dynamically rate the quality of an arbitrarily long video sequence using a slider mechanism with an associated quality scale. This gives us a higher sampling rate which can track rapid changes in quality, and is more useful for evaluating real-time systems. However, there are some negative aspects. SSCQE is susceptible to context effects. The scores can also drift over time if the subjects lose track of the absolute slider position on the rating scale, resulting in lower accuracy of the results. The method can be modified to attempt to get the same benefits as double stimulus methods by using hidden reference removal. Viewers will then get reference video mixed in with the impaired video without their knowledge, and the final score is modified based on this. When testing this in [25] it was found that reaction time became a factor, affecting the score accuracy.

2.5 Summary

This chapter gave an overview of the fundamentals of block-based video compression as used in the scalable video codecs mentioned (section 2.1). Scalable video (section 2.2) was defined as adjustments done to the video data in order to get graceful changes in quality, without the need for separate video streams. Stream switching appears as a much more coarse jump in quality and requires more time to encode and more storage space. Two scalable video codecs, SPEG and MPEG-4 FGS were described. The focus was on what techniques they used to attain scalability.

After the video section a description of the network metrics bandwidth, latency, jitter and packet loss followed (section 2.3), and how they might affect streaming video. The conclusion from that was that latency would have no effect on the tests in the project, and bandwidth and jitter would have the same effect as packet loss from a client side perspective. Finally an overview was given of common ways to measure image quality (section 2.4). Both objective and subjective image quality metrics were described.

Chapter 3

Test Method

Scaling a video stream on the sender side involves adjusting encoding parameters based on some feedback (for instance packet loss measurements), with resolution, bitrate and framerate being among the critical ones. The project needed to create a set of video clips covering a suitable range of parameter variations to test how those variations affected perceived quality. The source material for those clips had to cover a wide range of compression complexity due to a desire to create a general quality measure.

Network problems resulting in packet loss introduce other visual artifacts than the encoding process. Missing data will appear to the viewer as a different anomaly than what can come out of the compression process. Since they are of a different nature than encoding artifacts it would have been interesting to measure what kind of impact they had on perceived quality. Creating clips with this kind of impairment was planned, and a method for it was created, but it was dropped due to time and resource constraints. The method is still detailed in this chapter for completeness and future reference.

3.1 Video Clip Characteristics

The video clips used for testing were chosen based on the need for them to represent a wide range of scenes and their content. Error visibility is dependent on the quality of the source material as well as compression settings, and this was taken into account. Since all the chosen sources were from DVD, some material had been softened to reduce detail in the image, which raises compressibility. Scene complexity can be described by looking at spatial detail, temporal activity (motion), contrast range and color variety:

- *Spatial detail*. The amount of detail, or information, described by the pixels of an image. This is closely coupled with the resolution, since more pixels are required to describe finer detail.

- *Temporal activity*. This means the amount of motion a clip contains, where motion is defined as the difference from one consecutive frame to the next. Fast moving scenes, where either the contents of a frame as a whole moves (camera pan) or the objects within (car moving), will have more differences between frames than a slow moving or static scene.
- *Contrast range*. The difference between the highest and lowest luminance value in the clip.
- *Color variety*. The range of colors used for the pixel values in the images of the clip.

All of these elements require more bits to describe accurately during compression, depending on the degree/amount. They will all affect the complexity of a clip in the sense that it will be harder for a MPEG-4 video encoder to compress. It will need more bits to describe the scene or will have to forsake quality. The video clips selected for testing would, in total, try to cover the full range of encoding complexity.

3.2 Bitrates

Choosing bitrates for the clips depended on what kind of device and/or network the video was meant for. For clarity this thesis used three terms to describe technology segments:

- *Low-end*. This segment comprises small devices which typically have very low-bandwidth network connections. UMTS based mobile phones are good examples.
- *Mid-end*. Various DSL technologies fit well into this description, e.g. ADSL, SDSL, VDSL. In other words, typical home internet connections bandwidth wise, and common computers and display devices for the video.
- *High-end*. For DVD and higher (HDTV) quality, at least 10 Mbit/s is needed, which makes this segment only suitable for a LAN or equivalent internet connection. Additionally a suitable display is needed, for instance a HDTV capable TV typically larger than common PC displays.

A good example of low-end bitrates was the UMTS standard for 3G mobile phone networks [26]. The UMTS specification allowed the following data rates:

- 2.048Mb/s for pico-cell (and micro-cell) applications.
- 384kb/s for medium size cells (micro and small macro cells).
- 144kb/s and 64kb/s for large cell applications (large macro cells).

- 14.4kb/s for continuous low speed data applications in very large cells.
- 12.2kb/s for speech (4.75kb/s - 12.2kb/s).
- 9.6kb/s globally (satellite).

Since the bitrates should be usable on current technology, the basis for low bitrate clips was what the largest national phone carrier could provide. In Norway this was Telenor, and they had the following data rates for UMTS and EDGE [7] as of February 2006:

- UMTS: 384 kbit/s downstream, 64 kbit/s upstream.
- EDGE: 100-200 kbit/s downstream, 50-75 kbit/s upstream.

By assuming a conservative overhead estimate of 20%, which also included a small amount of additional traffic which might be in the background, the result was a $\tilde{3}$00 kbit/s bitrate for the low-bitrate segment. For video clips of that kind the audio was likely to lie within the 16-64 kbit/s range, and with that in mind the project chose a video bitrate of 240 kbit/s.

For the mid-end bitrate Telenor was once again used for guidance. They had the following ADSL products as of February 2006:

- Mini 700/160 kbps
- Basis 1500/300 kbps
- Pluss 3000/350 kbps
- Ekstra 6000/500 kbps

MPEG-4 video was designed to deliver DVD quality at lower data rates than MPEG-2. As mentioned later in this section, 4 Mbit/s is not an uncommon average for DVD material. With this in mind the project made the assumption that it would be possible to achieve visual quality comparable to DVD at bitrates beyond 2 Mbit/s with a MPEG-4 video codec. As the mid-end bitrate representative the Basis product was the one which seemed most suitable as it had adequate bandwidth to distinguish itself from the low-bitrate segment while not intruding upon the high-end segment. Without overhead (20%) the result was 1200 kbit/s. Audio bitrates were then estimated for these kind of clips to be 64-192 kbit/s, and so 1000 kbit/s was chosen for the video bitrate.

Next up was the high-end bitrate. A random sampling of average bitrates for 10 regular DVD releases showed that 4000 kbit/s was fairly representative. At this quality level the project aimed for full SDTV broadcast and home theater use. By using DVD as reference, despite also allowing MPEG audio layer II, DTS and PCM the de-facto standard audio codec is Dolby Digital at 192-448 kbit/s, depending on quality and amount of channels used. This would then result in $\tilde{3}$500 kbit/s for

video, which should be adequate for a very high quality image when using MPEG-4 ASP.

Out of the three proposed quality segments only the mid-range segment was selected for testing. Testing the other segments would have increased the time requirement substantially. Setting up a proper playback environment for them would also have been difficult. The mid-range segment was the most likely target for streaming video applications since it operated within the bitrate boundaries for common broadband connections. Devices in this range were also likely to be able to decode the video without any issues.

3.2.1 Resolutions

The resolution used for the test clips was directly tied to the bitrate. A higher resolution image had better spatial fidelity but needed more bits to conserve the added detail. Screen size was also a factor since downscaling the image would discard detail information, so a high resolution image did not make sense for small screens with low resolution. Upscaling did not necessarily impact the perceived image quality as much, depending on the scaling algorithm used (e.g. Bilinear, Bicubic, Lanczos, etc.).

The project used industry standard resolution definitions for maximum values. Some common ones are listed in fig. 3.1. Take note that QCIF, CIF and D1 are meant for television use and have non-square pixels. Since computer graphics hardware generally use square pixels, video clips using the native aspect ratio of these resolutions must be scaled properly on playback, or be cropped and scaled to fit the frame in the proper aspect ratio when displayed in a 1:1 pixel mapping.

The resolution for the low-bitrate clips had to fit the screen of typical hand-held devices, like mobile phones and PDAs. As the basis for our choice of resolution was the Sony Ericsson K750i GSM phone which had a 176x220 pixel TFT screen capable of displaying 262K colors. QCIF was a perfect choice for this device, and since it had a screen height larger than width it would also be possible to increase the resolution for widescreen material by viewing the clips sideways (rotated 90 degrees).

At the mid-end a typical computer display was considered to lie in the 15-19 inch range. Resolutions from CIF to D1 would be suitable. Typical screen aspect ratios were 4:3, 5:4, 16:9 and 16:10. The project decided to use a vertical size of 480 pixels as representative maximum resolution for this segment. The reasoning behind this was that for the bitrates in question the total pixel count should not be higher than the pixel count of the resolution 640x480. This is the highest common PC resolution that does not equal or exceed DVD.

The high-end targeted full DVD and HDTV resolution clips. For DVD this meant 720x576 for PAL material, and 720x480 for NTSC. HDTV enabled also using 1280x720 and 1920x1080 resolutions. Widescreen DVD movies are stored in an anamorphic format, where the image is squeezed horizontally to fit the frame while maintaining full vertical resolution at the cost of some spatial fidelity. Still,

QCIF	
176x144	(PAL)
176x120	(NTSC)
CIF	
352x288	(PAL)
352x240	(NTSC)
D1	
704x576	(TV PAL)
704x480	(TV NTSC)
720x576	(DVD-Video PAL)
720x480	(DVD-Video NTSC)
TV/PC 1:1	
768x576	(PAL)
640x480	(NTSC)

Figure 3.1: Various standard resolutions

this is preferable to letterboxing the image which essentially gives us a full aspect but downsized image. Anamorphic frames are scaled to the proper resolution and aspect upon playback, and are formatted to fit a 16:9 display in the case of DVD. HDTV on the other hand transmits full resolution video, which has been standardized to use a 16:9 aspect ratio.

The test clips were not stored as anamorphic video but were cropped and scaled to the proper source aspect ratio. A non-anamorphic image would be more representative for how digital video is stored for PC, Internet and high-definition use. Since the clips did not all have the same aspect ratio, a static height and total pixels per frame limit was used as shown in fig. 3.2.

Resolution	Original Aspect	Real Aspect	Pixels
Max:			
640x480	4:3	4:3	307200
Clips:			
592x320	1.85:1	1.850:1	189440
704x384	1.85:1	1.833:1	270336
640x272	2.35:1	2.353:1	174080
784x336	2.35:1	2.333:1	263424

Figure 3.2: Aspect ratios and total pixels per frame

3.2.2 Playback And Chroma Issues

Video stored in a compressed color space format like YV12 must often be converted to a different color space before being presented to the viewer. Since the chromatic component of such a signal is sampled at a lower resolution than the luminance, it has to be scaled. Simple scaling algorithms will display aliasing in the color parts of the picture, so care must be taken to avoid this so it will not be confused with stream errors and affect the judgment of a test participant.

In the test environment all clips used the YV12 color space, and the chroma conversion done by the display driver was judged to be of sufficient quality. This evaluation was done subjectively in a visual manner. For the highest quality a conversion to RGB with a high quality scaling algorithm could have been used (ffdshow [4] could do this). Without additional testing it was not known if the extra CPU performance hit would disrupt playback, so this configuration was not used for the project.

3.3 Test Environment

To create as realistic a test environment as possible the streams were created using tools implementing industry standards. The video was encoded with the Xvid codec [19] as MPEG-4 video using the Advanced Simple Profile (ASP). For streaming the project used Darwin Streaming Server [13] (DSS), which employs the protocols RTSP (RFC 2326) and RTP (RFC 3550). The streams were recorded on the receiving end by the openRTSP command line client from the live.com streaming tools project [15].

The server ran Mandriva Linux 2005SE with kernel v2.6.12 (as provided and patched by Mandriva), DSS v5.1.1, MPEG4IP v1.3 and iproute2 tools dated 10.01.2006. The client ran Microsoft Windows XP SP2, with openRTSP from the live.com library dated 05.01.2006. All encoding and primary file manipulation was done on the Windows platform due to it having a good selection of tools not available on other platforms.

3.3.1 Video Encoding

The source material was copied from DVD (PAL MPEG-2, 720x576 YUV 4:2:0, 25 fps) and then encoded to MPEG-4 ASP video with the tool MEncoder [31], a part of the MPlayer project. MEncoder could choose between two ASP capable video codecs, libavcodec and Xvid, and we chose Xvid for our clips. It was chosen for its configurability and consistent quality during initial testing. The thesis author has also had previous experience with it. Acquiring the source video from DVD and preparing it for processing required a number of steps:

- *Decryption.* Most DVDs are encrypted, but there are various methods of decrypting the content as documented on some internet forums [5]. The

decryption process also enabled extraction of video on a per-chapter basis, which allowed the project to work on smaller files which sped up the processing and reduced storage requirements.

- *Index*. This was the process of gathering information about the video stream, and was performed by DGIndex [10]. Frame number, size (in bytes) and frame type was written to a D2V file.

- *Cropping*. The video stream stored on DVD was formatted to fit within a standard D1 frame size (720x576 for PAL, 720x480 for NTSC). A full 4:3 (or 16:9 if anamorphic) image would fill the entire frame, but an image with a different aspect ratio, like 2.35:1, would have to be letterboxed with black bars to achieve the proper aspect. These bars decrease compression efficiency, although not significantly since they are mostly a uniform black. For internet and PC use it is more common to store only the video data without any black bars, and let all scaling and aspect ratio correction be done by the hardware or software performing playback. With this in mind the project had to remove the black bars in the source material before encoding. If the source material was frameserved by an AVISynth script, then MEncoder could be used to detect how an image had been letterboxed by using the `cropdetect` image filter. The output of this filter had the form `[width:height:x:y]`, where `x` and `y` determined the topleft start position of the actual picture, and `width` and `height` determined the size.

- *Find clip*. In this step a suitable place was found in the chapter to extract 10 seconds worth of frames. The VirtualDub [20] video editing software was used for frame-accurate browsing. Take note that this amounted to 249 frames and not 250, since also the last frame would be shown for its duration (40 ms in PAL), which would bring the total playtime to exactly 10 seconds.

- *Create AVS*. Reference video needs to be free of any encoding impairments and must usually be stored in a lossless manner because of this. Storing lossless video can quickly consume available storage space. Consider a 640x480 resolution video using the RGB color space with 8 bits per channel, and having a framerate of 25 fps. The bandwidth required would be $640 * 480 * 24 * 25 = 184,32$ Mbit/s. Instead of processing the reference video and storing it in a lossless manner, a method called "frame-serving" was used. This technique involved accessing the video through a processing script that external applications treated as a video file. The frames "served" from this fake video file were processed in real-time according to the commands in the script. AviSynth [29] enabled the project to use this method. The project created scripts detailing which part of the reference video to be used in addition to what processing to be done, i.e. cropping, scaling, framerate conversion, etc. Separate scripts were written for the various resolution

and framerate variations to be tested. An example script can be seen in fig. 3.3.

```
# Needed for D2V file support
LoadPlugin(‘‘DGDecode.dll’’)
MPEG2Source(‘‘daggers.d2v’’)
# Choose a range of frames
Trim(4078,4326)
# Reduce framerate to 15 (drop frames)
ChangeFPS(15.0)
# Set 15 fps as new framerate
AssumeFPS(15.0)
# Remove black borders
Crop(0,82,720,416)
# Resize
LanczosResize(640,272)
```

Figure 3.3: Example AVS script

MPEG video has strong dependencies between frames, causing errors to propagate temporally. This is seen in P-frames, which describe changes at some point in time from the previous reference frame (I or P), as well as in B-frames, which depend on both the previous and next reference frame (I or P). Depending on the type of stream used some errors can be removed through error correction. Since stream overhead usually is an issue this will only work for minor problems. Error concealment is an option for what error correction cannot handle, for instance by finding the missing data through interpolation. Regardless of method used, at some point a perceived impairment will propagate temporally and can only be stopped by introducing a new intra coded block for that part of the image. One way of minimizing the distance between intra coded data, and hence how long an impairment stays in the image, is to make sure I-frames are coded in the stream at regular intervals. In MPEG video the maximum GOP (Group Of Pictures) size determines how long a sequence of P- and B-frames after an I-frame can be before a new I-frame is inserted. Setting a small GOP size ensures a maximum distance between I-frames. For the created video clips the project did not use a fixed frame pattern for the GOP (which is typical for broadcasting and DVD), but instead let the encoder decide the best approach. The maximum GOP was limited to 2 seconds (50 frames for PAL).

MEncoder supported high configurability of Xvid encoding options, but bitrate and max_key_interval were the only relevant parameters. The rest were pure quality parameters which traded encoding time for visual quality, but the test clips were all encoded with maximum quality settings as the time taken was found to not be prohibitive. A detailed description of these parameters are beyond the scope of the thesis. Refer to the Xvid [19] mailing list, the Xvid source code, and the Doom9

forum [5] for more information.

3.3.2 MPEG-4 File Creation

The MPEG-4 video elementary streams were put in MPEG-4 container files using MP4Box from the GPAC MPEG-4 Systems multimedia framework [8]. An alternative, though less feature rich, were the MPEG4IP [22] tools. It was the `mp4creator` tool from that package that was used for hinting. Hinting a track added a control data track which the streaming server could use to optimize data delivery, and DSS required this. This track was only used by the server and was not sent over the network. Optionally the files could also have been optimized. This process allowed better HTTP streaming by interleaving the tracks contained in the file in time, and by moving media control information to the front of it. Since the test clips contained a single track and were only meant for RTP streaming, this step was skipped.

Using MP4 files avoided an issue with AVI parsers and out-of-order frames like B-frames. In a MPEG stream there are two types of timestamps, decoding timestamps (DTS) and presentation timestamps (PTS). The former indicates when, and hence in what order, a frame or group of frames should be decoded, while the latter tells the parser when the frame or frames should be presented to the display device. AVI was designed for decoding frames in a linear fashion without delay, so in AVI there is only one timestamp meaning the DTS is equal to the PTS. This means there is nothing inherently wrong with using B-frames in AVI, but it becomes the parsers job to deliver frames in the proper order to the decoder and there are many parsers that fail at this. DivX [14] created a solution for this called packed bitstream, where several frames are packed together to appear as one AVI frame to give the appearance of a linear frame order, but once again parsers must be aware of this method for it to work.

All test clips were accessed through AviSynth scripts using the DirectShowSource method. This raised one concern for other usage areas besides those detailed for the project. A DirectShow [2] parser filter is not required to be frame accurate when performing seeking in a stream. For the project this was not an issue since the video clips were always played from beginning to end. However, if an evaluation between two encodings of the same video clip was needed, comparing frames directly would be problematic since seeking to a frame would not be guaranteed to deliver the correct one. Testing revealed that the MP4 parser filter chosen, Haali Media Splitter [23], did not provide frame accurate seeking.

3.3.3 Streaming Server

Darwin Streaming Server from Apple had several properties that made it suitable for the test environment:

- Since it serves as the basis for the commercial Quicktime Streaming Server

the code can be classified as production ready and not experimental. QSS/DSS also has a sizeable market penetration, giving the tests a realistic flair.

- It is standards compliant when it comes to RTSP/RTP streaming using MPEG-4 files and MPEG audio/video content, though not necessarily feature complete when it comes to optional parts of the standards. For instance, DSS does not support all available RTP payload formats.
- The server is designed to be cross-platform software, enabling flexibility in the design of a test environment. This also makes the environment easier to reproduce.

DSS did not need any special configuration options aside from the initial setup. Media files were placed under a user-configured root directory, and accessed by using an URL of the kind `rtsp://server/file.mp4`.

3.3.4 Stream Recording

To record the video streams the project used the command line RTSP client openRTSP [15]. In the version of openRTSP used, 2006-01-05, MP4 multiplexing had issues and produced files that stuttered on playback, so the command line used (fig. 3.4) stored the data in a MPEG-4 video elementary stream which was multiplexed to MP4 using MP4Box. The structure of the new file was very close to the original, and the video stream it contained remained bit identical.

```
openRTSP -v -b 65535 rtsp://10.0.0.100/file.mp4
 > file.m4v
```

Figure 3.4: openRTSP command line

3.3.5 Network Emulation

For network emulation the project used the NetEm [12] QoS filter in the Linux kernel. It provided the necessary tools to emulate delay, loss, duplication and packet reordering. Bandwidth was restricted by using the Token Bucket filter. NetEm was controlled with the `tc` tool from the `iproute2` [11] package. An example that sets a bandwidth limit and a loss rate can be found in fig. 3.5. Take note that latency in the Token Bucket filter means an upper limit for how long a packet can stay in the queue, and does not impose a delay on all packets passing through.

3.3.6 Network Factors

Important network parameters for streaming video experiments are bandwidth, latency, jitter and packet loss. Which parameters to choose for the tests depended on the effect variations in them would have on perceived video quality.

```
tc qdisc add dev eth0 root handle 1:  tbf
 rate 1024kbit latency 100ms burst 15400
tc qdisc add dev eth0 parent 1:1 handle
 10:  netem loss .1%
```

Figure 3.5: Example `tc` command

- *Bandwidth*. The streaming server used did not do real-time encoding, and had no means of scalability by controlling bandwidth usage based on network status. Hence the video clips would be encoded at a fixed bitrate to fit within a predefined bandwidth. If the bandwidth was lowered due to congestion or some other network event, packets might have been discarded or delayed at various points along the network path, which would have the same effect as packet loss to the client. Choosing bandwidth as a test parameter would then be redundant if packet loss was included.

- *Latency*. Since the streams were dumped to file for playback at a later time, a fixed latency would only delay arrival of the stream. It would not affect the quality of the recorded data, so latency was not included as a test parameter.

- *Jitter*. The streams were stored in files and had the same length and timestamps as the version stored on the server side. Variations in packet latency would not affect the file structure itself unless the client side buffer was very small. Jitter could then cause data loss if the buffer became empty or packets timed out, but that would have had the same client-side effect as packet loss. To include jitter in the tests would then be redundant.

- *Packet loss*. This network parameter would have the most impact on the video streams. Loss of packets would mean missing data, which in turn would lead to various types of image corruption. With bandwidth and jitter having the same client-side effect as packet loss, and latency not being important for the tests, packet loss became the only network parameter of relevance.

Having defined packet loss as a primary factor, the project researched which parameter values would be most descriptive for the tests. Some packet loss characteristics were described for certain kinds of networks in the literature. In [3] the authors found an error rate of 10^{-2} acceptable for speech and 10^{-5} acceptable for uncoded data. For fiber optic networks, [17] claimed an error rate of at most 10^{-7} was acceptable in order to sustain reasonable transfer rates. However, with loss rates that low the image quality degradation would be mostly imperceptible for the amount of packets needed to transfer 10 seconds worth of data. On the other hand, a loss rate that was too high would cause the image to be distorted beyond recognition. The project needed to perform tests to find suitable upper and lower limits,

and then test several steps between. These tests were not actually performed since packet loss was discarded as a test parameter due to the parameter combination matrix (fig. 3.7) becoming too large for the time available.

3.4 Parameter Matrix

Having settled on some parameter variations to make video encodings from, a full combination matrix (fig. 3.6 and fig. 3.7) was created. There was to be one encoding representing each possible combination. With two variations for resolution, two for framerate, and three for bitrate, the total combinations were 12 for each test clip.

	FPS_{15}	FPS_{25}
RES_{med}		
RES_{high}		

Figure 3.6: Resolution and framerate matrix

	BR_{500}	BR_{1000}	BR_{2000}
$RES_{med} * FPS_{15}$			
$RES_{med} * FPS_{25}$			
$RES_{high} * FPS_{15}$			
$RES_{high} * FPS_{25}$			

Figure 3.7: Resolution, framerate and bitrate matrix

3.5 Objective Evaluation

All objective tests were performed with [34]. This tool supported a wide range of metrics, and the project chose to use PSNR, SSIM and VQM for the following reasons:

- *PSNR*. Although not useful in judging how impairments were perceived, it was still a commonly used metric and would at least give an adequate measurement for the degree of error in the mathematical sense.

- *SSIM*. Since this metric tried to emulate one of the later stages of the HVS, namely structure identification, it was suitable for representing HVS based methods. It was also a relatively new metric with a promising background and test results [37].
- *VQM*. Using only one HVS based metric eliminated the possibility of verification, so VQM results were also added. It was at the time also the only other HVS based metric available in the test suite. This metric emulated the early stages of the HVS by using the DCT to measure error in the frequency domain, while taking into account the frequency sensitivity of the human eye.

The resulting data from the tests was stored in files having the kind of structure seen in fig. 3.8. In addition the option to create videos showing the difference between the reference and the tested clip was also used. These videos act as visual representations of the errors as reported by the metrics, and were helpful to locate them and see how severe they were. They were also great for presentation purposes to gain an understanding of video impairments, and as training to locate them. PSNR videos use color coding (from no to high severity: black, blue, green, yellow, red) on a per pixel basis to indicate the degree of error. SSIM videos are grayscale and use stronger luminance values to indicate a higher degree of error. VQM evaluates 8x8 pixel blocks at a time, and the difference videos show these blocks in grayscale from black to white, from no to high degree of error respectively.

```
PSNRYYUV
D:\path\to\daggers_640x272_fps15_br500.avs
38.3024
35.7618
36.2077
...
AVG: 33.4108
```

Figure 3.8: Example data file, PSNR test

The project needed a way to view per-frame statistics, as well as the average, median and min/max values. Finally, scripts had to be made to plot graphs with gnuplot [39]. To accomplish all of this a Python script was made, makedat.py C.1. Running the script in the folder containing the CVS files produced a `plot` directory with plot script files, and a subdirectory `dat` containing gnuplot compatible data files. The files in the `dat` directory with the same name as the CSV file it was derived from contained two columns, where the first was the frame number and the second was the test score for that frame (fig. 3.9).

The other files contained the average and median score, and had names ending with `_avg` and `_med` respectively. Those files had a single entry with the first

```
# PSNRYYUV
# daggers_640x272_fps15_br500.avs
0 38.3024
1 35.7618
2 36.2077
...
```

Figure 3.9: DAT file example

column being the clip ID (as defined by the `makedat.py` script), and the second column being the average or median value. The ID was needed for showing several results together in comparison plots. Finally, the script generated plot files to automate gnuplot, which in turn created both a PNG image for computer use and an encapsulated postscript (EPS) file for print usage. The per-frame statistics were plotted as line graphs with the frame number on the x-axis and the test score on the y-axis (example fig. B.1). The average and median graphs use error-bars to illustrate the max/min values for each clip, and where on that bar the average or median lies (example fig. B.5). Running a simple `gnuplot *.plot` in the same directory as the plot files generated all the graphs.

3.6 Subjective Evaluation

For the subjective tests the project could choose between several single stimulus and double stimulus metrics, and the project chose Double Stimulus Continuous Quality Scale (DSCQS). This choice was based on the properties of DSCQS and the fact that it has shown itself to be a reliable test method [30]. Two tools supporting this metric were tested: The MSU Perceptual Video Quality Tool [35] and Video Quality Studio [36]. Unfortunately, they were both judged as not being flexible enough. What the project needed was a way to evenly distribute the evaluation over the test data set by randomizing all steps involving choice of clips or ways of ordering them. The goal was to present to the test participant a sequence of pairs where each pair was a unique test clip. To gain full randomization the sequence order was to be random as well as the order of the clips within each pair. In addition, the impaired version of a clip chosen for a pair was to be random. All test participants would then have a high probability of getting a different viewing experience, which would act as compensation for eventual context- and memory-effects.

There was some confusion as to how the clip pairs should be presented. The MSU tool allowed two modes: Showing both clips in the pair at the same time, or to have them play simultaneously but only showing one at a time. In the latter mode the viewer could switch between the clips manually. VQ Studio could show clips sequentially or simultaneously, the former being the more traditional method. The project chose sequential viewing since simultaneous viewing would be affected by

memory effects, and would make it easier to reveal visual artifacts by its direct comparison nature. Also, blind testing is traditionally performed by being exposed to first one effect and then the other, and evaluating the perceived difference.

Since no suitable tools were found the project created one: `subjective.py` (C.4). This software was written in Python [33] and used the wxPython GUI library [6]. It was dependent upon some external tools to function: AviSynth [29], ffdshow [4] and Media Player Classic [9]. Upon starting the software the clip to view was randomly chosen, and a pair consisting of the reference and a randomly chosen impaired clip was generated. The subjects started the evaluation by pressing a button, and were shown the clip pair twice. After the initial viewing the subject could optionally view the clip pair again or proceed to set a score.

During development of the software some initial testing was done with five test subjects, and the option for subsequent viewings was included as a result of their feedback. All of them felt that the two initial viewings with ten second long clips was not enough to get a general impression of the video quality. Of course, this should have been tested by comparing two tests where one was restricted to only two viewings and the other not, but that was not possible for this thesis. It was decided that a more critically judged evaluation was better than an inaccurate one, given our expected small number of test participants.

After viewing the clip pair a number of times the test participants evaluated the quality of both clips using a scale from 1 to 5, from worst to best quality respectively. The difference between these scores were used as a measure for the perceived difference between the clips, as described in the DSCQS specification. After evaluation the participants went through the same steps for the remaining video clips.

The test PC had an Intel P4 2.6 GHz CPU, 1 GB RAM and a ATI Radeon 9600 graphics card. The display was a 19 inch LCD monitor, which was not calibrated using appropriate tools but adjusted manually based on subjective opinion. The room used had dim lighting, and had no strong outside interference audially or visually. Test participants were not trained in any specific way before the test, but since it was possible to view a clip pair multiple times they were instructed to set a score as soon as they had a general impression of the quality difference. Overly critical results could be avoided this way.

3.7 Video Clips

The test clips were selected based on the criteria mentioned earlier in the Video Clip Characteristics section. There were six chosen in total. The first four in the following list were used for the subjective tests and the last two were reserved for verification or other use.

Gladiator (Chapter 19)

The source for this clip was from the SuperBit Collection by Columbia-TriStar. DVDs in this collection have little overhead with regards to extra content besides the main movie. As much of the storage space as possible is allocated to the video track, and usually nothing else is included besides two audio tracks (Dolby Digital and DTS) and subtitles. The increase in video bitrate compared with typical DVD releases translates directly to improved picture quality in all aspects, especially concerning image detail.

The scenes in the clip had all been filmed with a static camera. Large parts of the image remained motionless. There were three scene cuts, which were not perceived as fast paced. Colors consisted mostly of brown tones. The image had in general a high detail level.

House of Flying Daggers (Chapter 3)

This source was not as detailed as a SuperBit DVD, but was still above average. There was no camera movement in this clip either. Rapid cuts with fast motion open the clip, which was reported as disorienting by some participants. Colors were vibrant and varied. The image suffered a bit from a flawed film transfer, with small (a pixel or two) movements of the whole image. Some test participants perceived that as an impairment.

Princess Mononoke (Chapter 8)

Also a static camera clip, it had two scene cuts placed far enough apart for the clip to be observed as having a slow pacing. Animated content often uses a mixture of full and reduced framerate (during creation, playback is full framerate). This was seen in the first scene, where the trees move at full rate but the characters obviously have been animated at a lesser rate. Reduced temporal fidelity should compress better, but changes from one frame to the next may be larger than usual. Colors were good, and detail was low compared to natural images. Large parts of the image consisted of flat color surfaces and sharp edges.

Resident Evil (Chapter 3)

This clip was the only one selected for testing that had camera movement. There was one scene cut, with camera zooming out and rotating in the second scene. The source was a good film transfer but had a “soft” image with below average detail. This was probably done to increase compressibility while retaining good image quality. Colors were not varied and consisted mostly of skin tones. There was a fair amount of detail in the closeup of the eye in the first scene.

Crouching Tiger, Hidden Dragon (Chapter 7)

This source was a SuperBit DVD, and had generally excellent image quality. The clip depicted a fast paced fight scene at night with much movement. There was little color variety, mostly blue and black tones. Detail level was high, but being a night scene it was difficult to observe.

Dracula (Chapter 6)

Also a SuperBit DVD, the image quality was very good. There was one scene cut. The first scene had a panning camera movement, the second used a static camera. Colors were vibrant and varied, and the image was very detailed.

Chapter 4

Results

Unfortunately, the subjective test method was flawed regarding the total number of participants. With only 15 subjects the choice of four test clips was a bad one, leaving the project with too few results per clip to reason about them. Even when merging the data to gain a general measure there were too few data points to say anything conclusive. Expanding the test session for each subject to include more clips, preferably all, would have been the proper choice, but there was no time to do so. Longer sequences would also have to account for the possibility of viewer fatigue [16, 25]. Still, based on both objective and subjective results the project could estimate how a utility function for perceived quality would look like.

For this section it became convenient to introduce the concepts of bits per frame (BPF) and bits per pixel (BPP). Bitrate determined the bits used for a number of frames in a certain amount of time, but did not directly say how the bits were distributed. For frame by frame comparisons BPF was a more natural measure. Objective tests work by measuring individual frames, so a comparison with them also became easier. BPP was useful when discussing the quality impact of different resolutions.

4.1 Objective Test Results

The primary focus of this thesis was not on objective tests, but they could still be used to reason about the subjective test results. By assuming the HVS based objective tests were a fair indication of subjective video quality despite not taking into account temporal effects, their precision could be estimated by comparing with the subjective test results. All of the objective tests were useful for estimating what kind of quality measurement function was needed.

First, some general observations regarding the test data before the individual test analysis. Since the objective tests evaluated individual frames, the clips with a 15 fps framerate got a higher score than the ones with 25 fps due to better BPF. For some clips it was clear that the quality dropped in a non-linear fashion as the bitrate got lower, and the situation was reversed as the bitrate went up. All graphs

showing a per frame test score had oscillation tendencies (look at fig. 4.1 for an example). This was caused by the compression of the frames in the clips not being the same due to using MPEG encoding, or more specifically, by using B-frames. MPEG video encoders typically encode these frames with higher quantizer values than I- and P-frames. This will usually not impact perceived quality much since B-frames can only reference I- and P-frames, which use lower quantizer values and hence have a higher image quality. Data referenced by a B-frame will then be of higher quality than what is stored in the frame itself in most cases. Due to using two frame references a B-frame is likely to contain more motion compensation data than other frame types instead of intra-coded blocks. Also, the lower quality data in a B-frame does not propagate in the stream since that data isn't referenced by other frames.

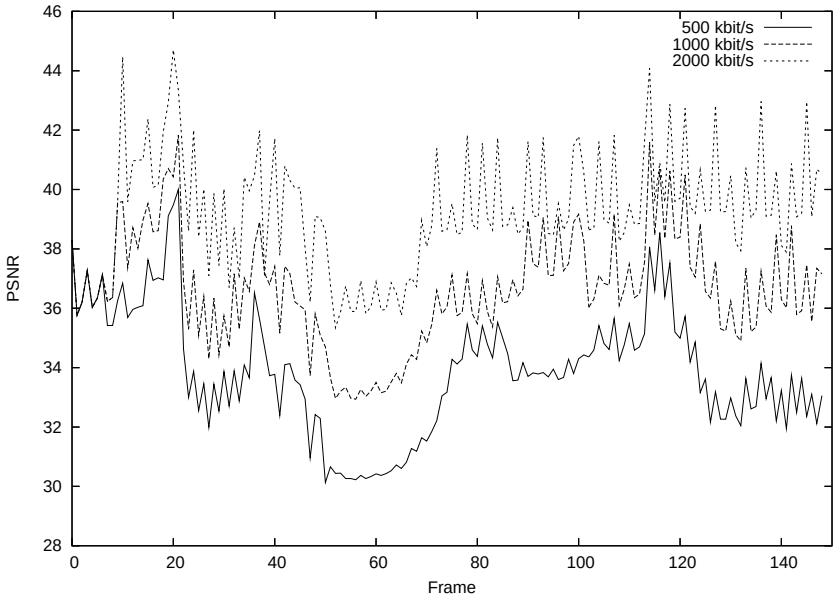

Figure 4.1: PSNR, daggers, 640x272, 15 fps

4.1.1 PSNR Results

A complete listing of all graphs from the PSNR tests can be found in appendix B.1.

daggers

This clip seemed to have high compression complexity and apparently did not saturate the bitrate based on two observations. First, the PSNR score for the medium resolution version was almost linear as bitrate increased (fig. 4.2). Second, the high resolution version had a slightly lower score (as expected due to BPF), but

retained the almost-linear growth (fig. 4.3). If bitrate was beginning to become saturated the score would have begun to level off.

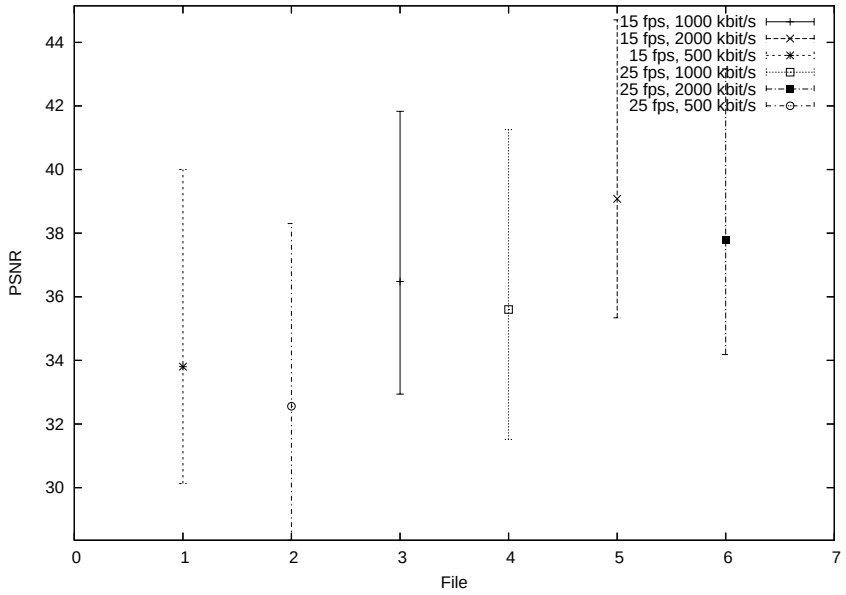

Figure 4.2: PSNR, daggers, 640x272, min/max/median

evil

The PSNR score for this clip revealed that it had low compression complexity. All clip versions had a score in close proximity to each other, meaning the bitrates tested were saturated. Only the version with the highest image parameter settings (high resolution, 25 fps) showed signs of diverging results at the different bitrates (fig. 4.4). This degree of compressibility was most likely caused by the source having a “soft” image with reduced spatial detail, and the scene displayed had limited color and structure variety.

gladiator

This was the first clip in this test where small signs could be seen of exponential quality development (fig. 4.5). Taken from a detailed source the compression complexity should have been relatively high. The resolution and framerate variations did not incur much of a PSNR score difference, but bitrate had a minor effect. As the difference between 1000 and 2000 kbit/s is smaller than between 500 and 1000 kbit/s despite having a larger bitrate increase, it can be assumed that the benefit from increasing the bitrate is getting lower at that level.

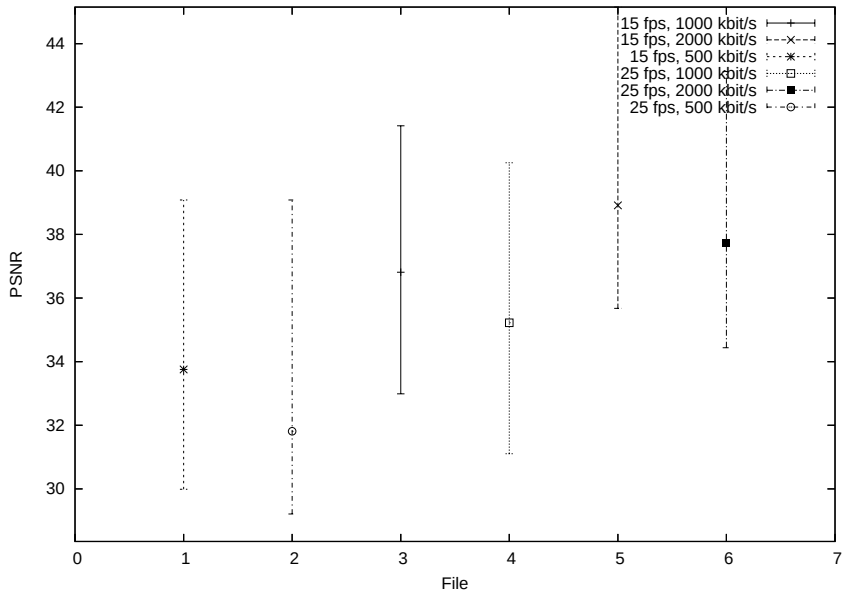

Figure 4.3: PSNR, daggers, 784x336, min/max/median

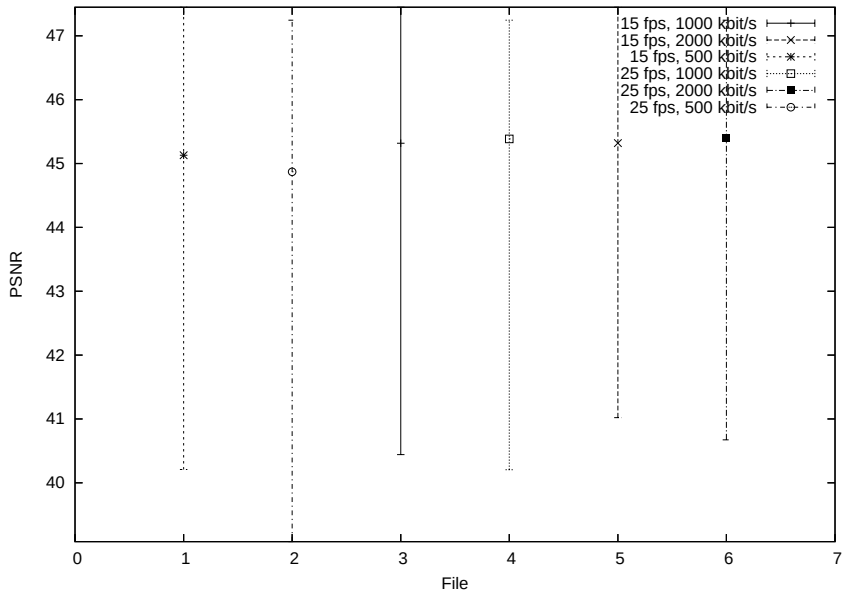

Figure 4.4: PSNR, evil, 704x384, min/max/median

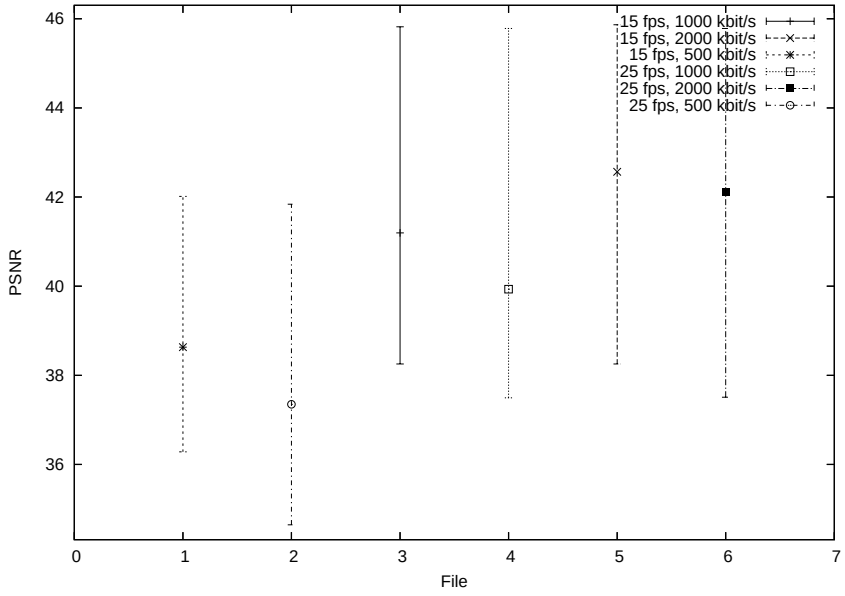

Figure 4.5: PSNR, gladiator, 640x272, min/max/median

princess

For the medium resolution, at both 15 and 25 fps the median only had small differences at 1000 and 2000 kbit/s, which were bitrate saturation symptoms (fig. 4.6). In the results for this clip it was even more apparent that the PSNR quality followed a non-linear growth (fig. 4.7). The effect of the framerate was small but noticeable, despite the source being animation which typically has smaller difference between frames than normal video material. An anomaly was the score for 25 fps at 2000 kbit/s which was higher than 15 fps, even though BPF was lower. The reason for this might be that the data files somehow have been swapped, but the two tests in question would have to be run again to confirm it.

4.1.2 SSIM Results

A complete listing of all graphs describing the SSIM results can be found in appendix B.2. For this metric higher score is better with 1.0 being the maximum.

daggers

SSIM measures structural difference, and due to the detail complexity of this clip it was possible to observe a noticeable difference with all parameter settings. Increasing the frames per second, and thus lowering BPF, had an impact at 500 kbit/s (fig. 4.8). Increasing the resolution had only a minor effect except at the lowest

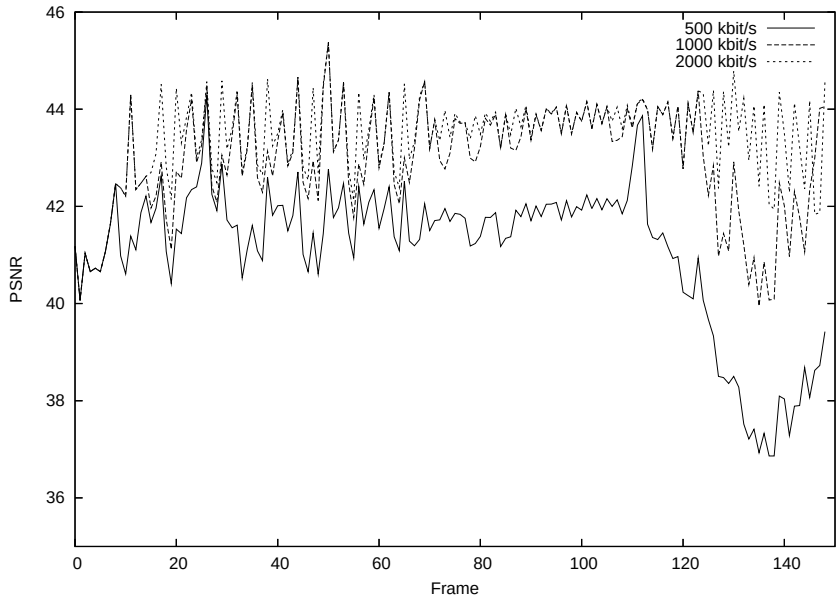

Figure 4.6: PSNR, princess, 704x384, 15 fps

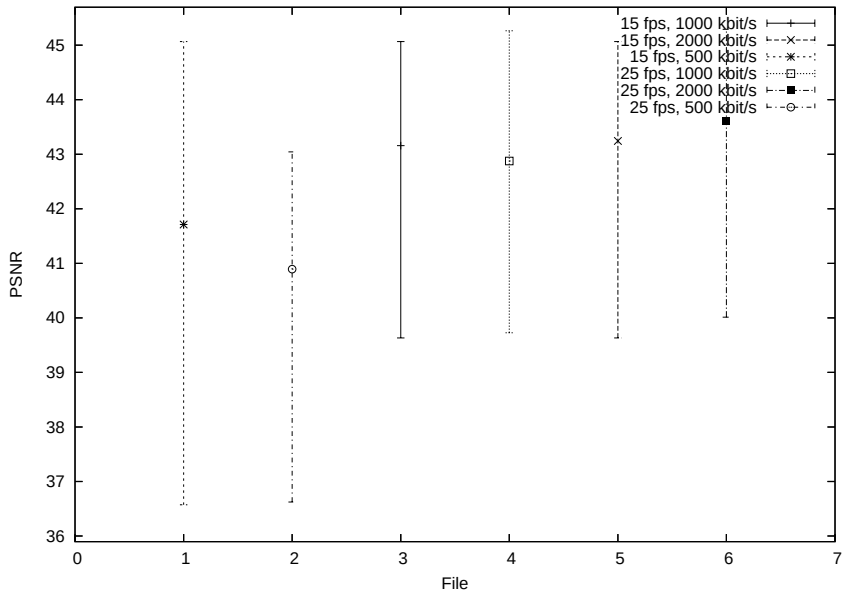

Figure 4.7: PSNR, princess, 592x320, min/max/median

bitrate. Comparing with the medium resolution median (fig. 4.9) showed that only 500 kbit/s could be considered a real jump in quality.

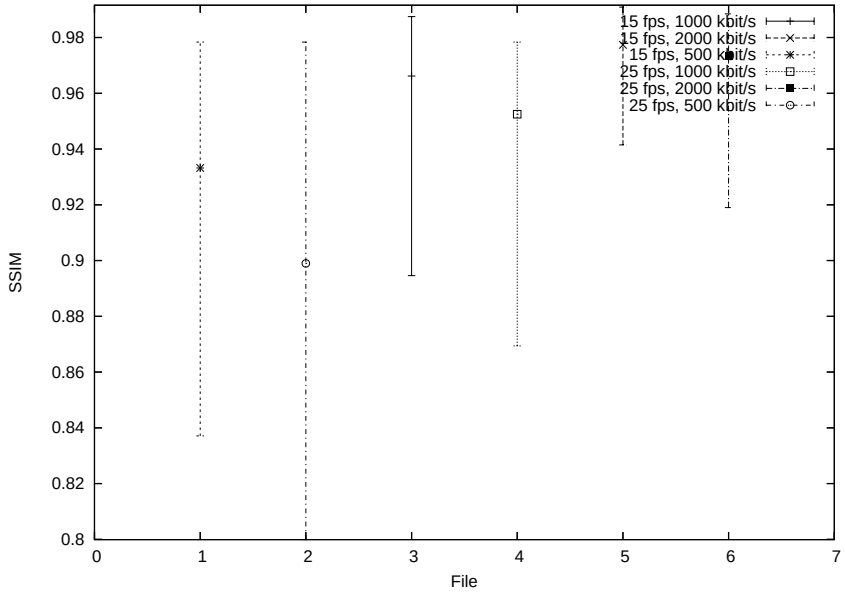

Figure 4.8: SSIM, daggers, 784x336, min/max/median

evil

Despite measuring different aspects of the video, both the PSNR and SSIM tests of this clip showed the same tendencies. Compression complexity remained low also for this test, resulting in small differences between the encodings (fig. 4.10). Based on the similarity of the scores, the overall quality could be considered to be high for all parameter variations.

gladiator

Based on the data, it would seem the difference between the tested resolutions were not enough to give significant structural difference (fig. 4.11 and fig. 4.12). Bitrate and framerate had some impact on the score, but overall the differences could be considered small.

princess

This animated clip scored well in general for this test. Only the 500 kbit/s clips had any impact, but even then the score could be considered high (fig. 4.13). Bitrate and framerate could not be said to influence the SSIM score much at 1000 and 2000 kbit/s.

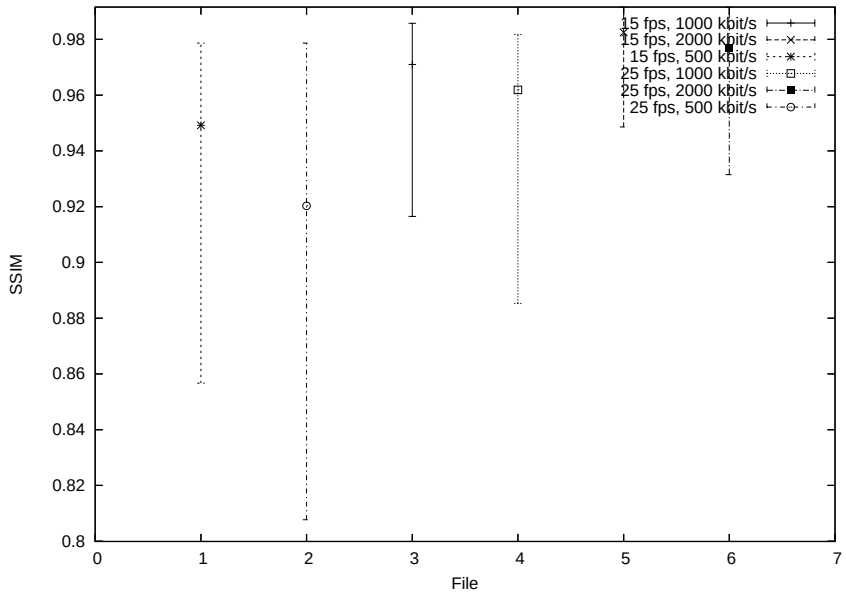

Figure 4.9: SSIM, daggers, 640x272, min/max/median

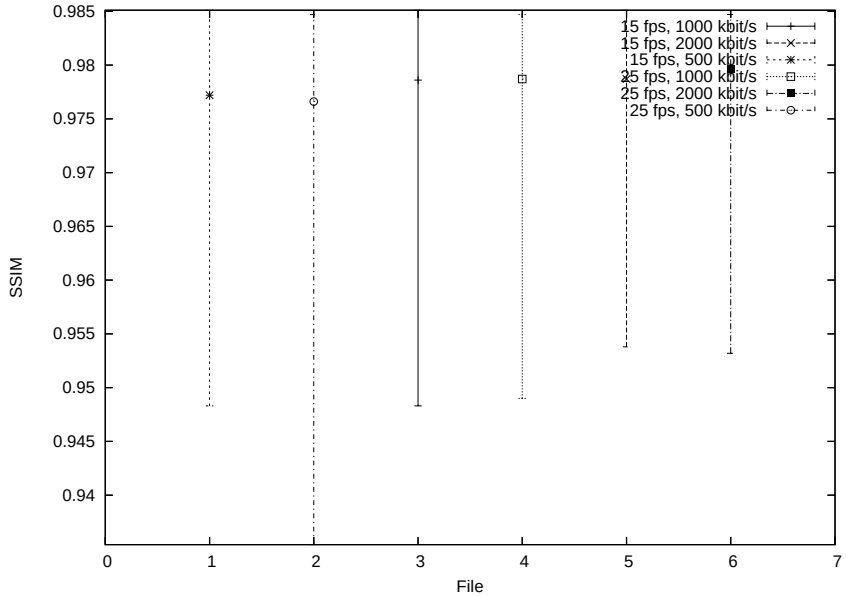

Figure 4.10: SSIM, evil, 704x384, min/max/median

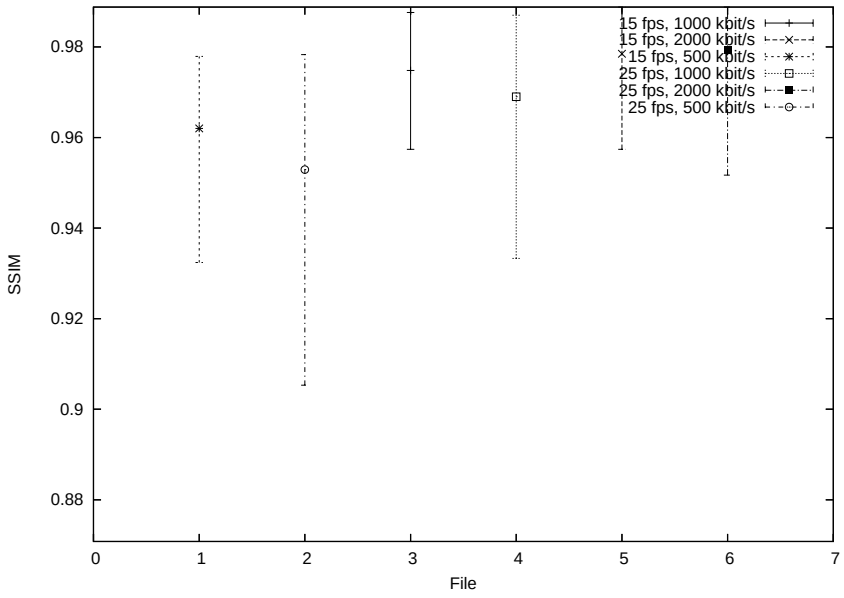

Figure 4.11: SSIM, gladiator, 640x272, min/max/median

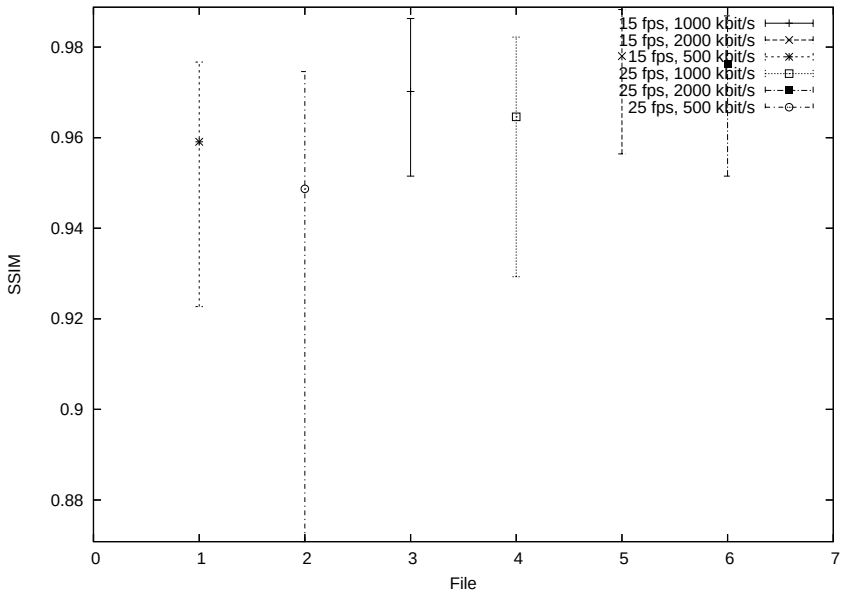

Figure 4.12: SSIM, gladiator, 784x336, min/max/median

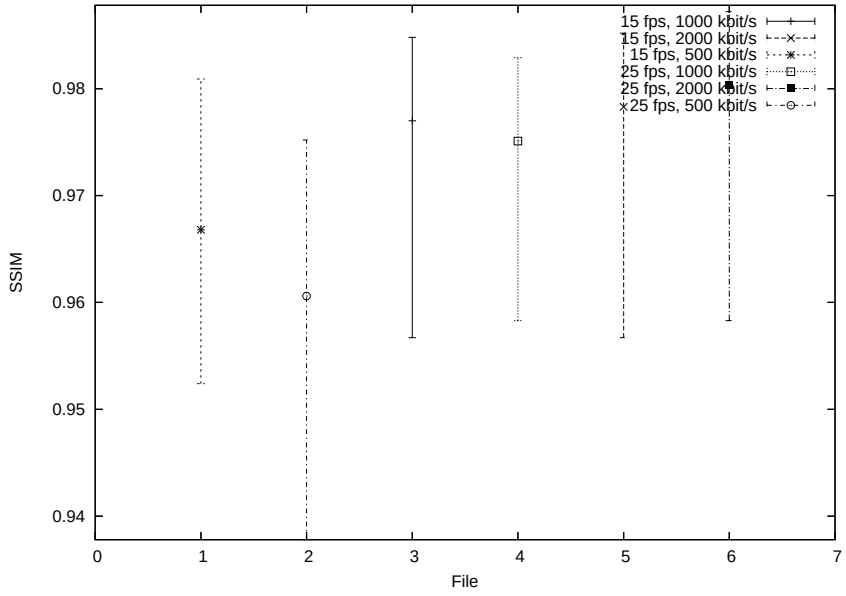

Figure 4.13: SSIM, princess, 704x384, min/max/median

4.1.3 VQM Results

For a full listing of the test result graphs, look in appendix B.3. This metric measures the amount of distortion from the reference, so lower score is better with 0 as the baseline.

daggers

This clip had close to linear degradation of distortion as bitrate rose (fig. 4.14 and fig. 4.15). Framerate could be seen to affect the score to some degree, but resolution had little impact. The clip was not being bitrate saturated at the tested parameter settings.

evil

Once again this clip was hampered by its low compression complexity (fig. 4.16). All parameter variations performed well, produced very similar results, and could be considered to yield a very low amount of distortion. According to the results the bitrates tested were all sufficient to produce a high quality image.

gladiator

This clip had good detail level, and this was reflected in the VQM scores. There was a noticeable difference between the scores for the parameter variations (fig.

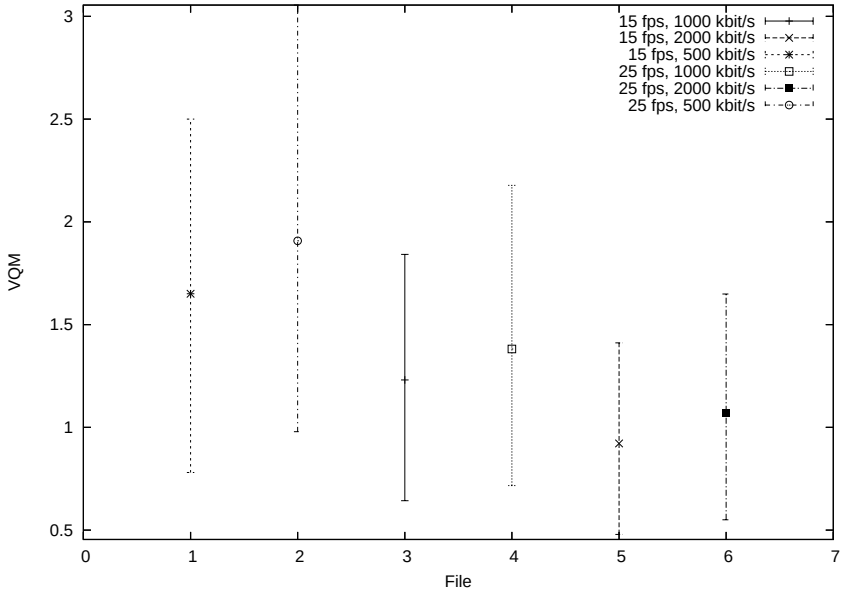

Figure 4.14: VQM, daggers, 640x272, min/max/median

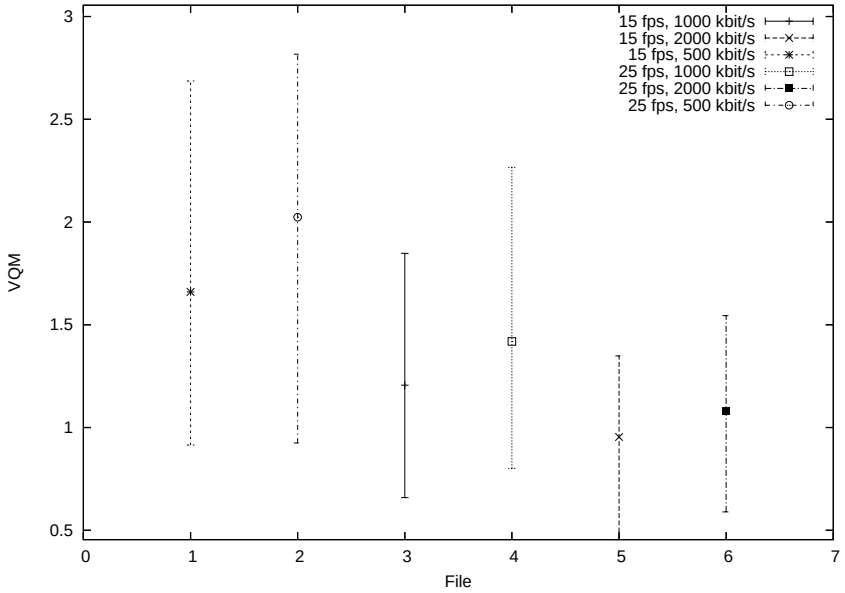

Figure 4.15: VQM, daggers, 784x336, min/max/median

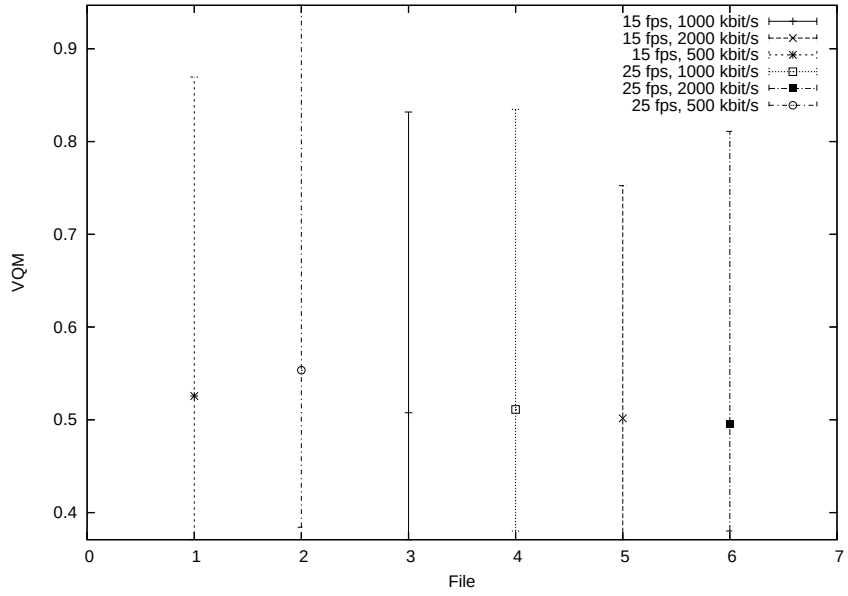

Figure 4.16: VQM, evil, 704x384, min/max/median

4.17), but the difference between the smallest and largest median value was relatively small. Telling the clip versions apart might have been difficult.

princess

As with the SSIM test for this clip, all versions except those with 500 kbit/s bitrate got similar results (fig. 4.18). Bitrate and framerate did not matter much. The distance between the scores was small, which showed that this clip had low compression complexity at the chosen parameter values and that it was getting bitrate saturated.

4.1.4 Summary

To summarize, only two clips, *daggers* and *gladiator*, seemed to have high enough compression complexity to not get bitrate saturated at the chosen parameter settings. Among the other two, *evil* in particular stood out as being highly compressible, with barely any quality change as the parameters varied. The *princess* clip had the same tendencies, but in a lesser manner. This fact would be interesting to compare with the subjective test results. If the viewers did not see much difference either then the result could be used, after more extensive testing, as verification of the accuracy of the HVS models employed in SSIM and VQM. It would also be possible to see if the objectively measured quality difference corellated with the subjectively perceived difference. This could be beneficial as subjective testing is

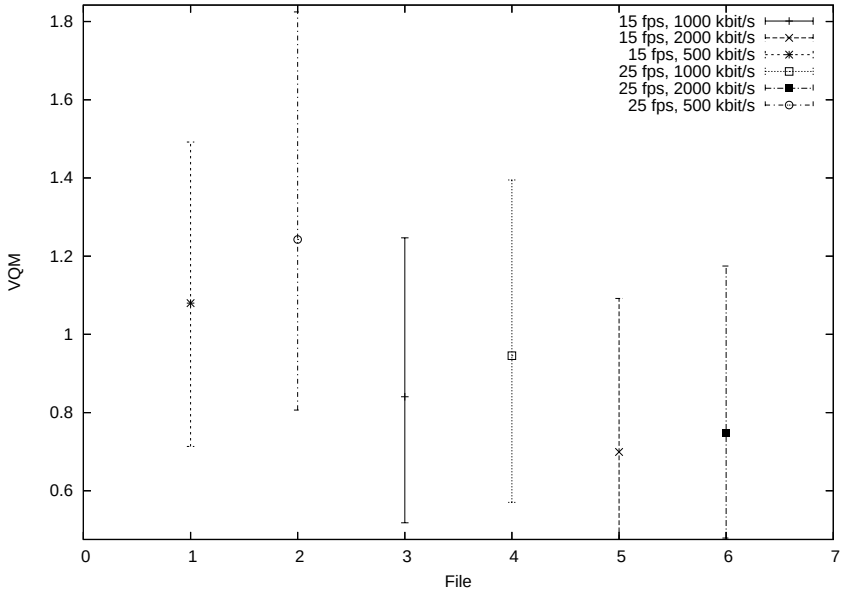

Figure 4.17: VQM, gladiator, 784x336, min/max/median

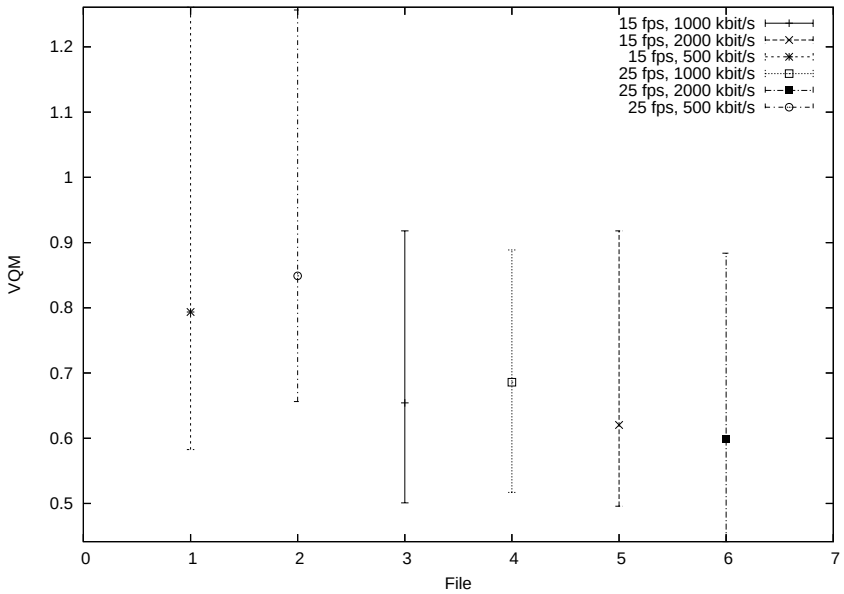

Figure 4.18: VQM, princess, 704x384, min/max/median

time and resource expensive, while objective testing can be considerably “cheaper” by being faster and not requiring actual people for evaluation.

4.2 Subjective Test Results

The project was unfortunately not able to perform a large scale subjective test, which would have been necessary for accurate statistics. Only 15 participants were tested, and with a data set that consisted of six variations for four unique video clips each, it followed that the gathered data on a per clip basis was not descriptive enough to draw any conclusions. Other tests have had a considerable higher number of test subjects [30] to get reliable results. However, the data was still usable, although crippled, by making the following assumption: Since it was to be the basis for a utility function describing a general purpose video quality metric, the global effect on as many scene types as possible was the ultimate goal. Individual clip statistics were useful to optimize the quality trade-off based on scene type, e.g. if it contained fast cuts, animation, or little color variety. They were not necessary to reason about the general effect of the impairments.

The subjective test results for the clips were merged, creating a data set with a higher statistical resolution. Two error-bar graphs were plotted, one for each resolution class (fig. 4.19 and fig. 4.20), showing min, max and median DSCQS score for each clip type. The 15 fps clips were presented first, in rising bitrate order, and then the 25 fps versions.

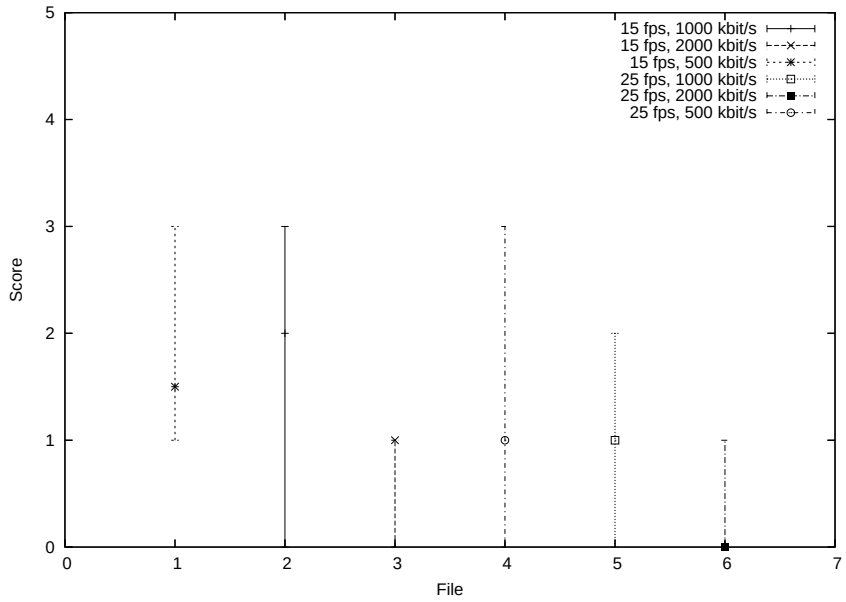

Figure 4.19: Medium resolution, min/max/median

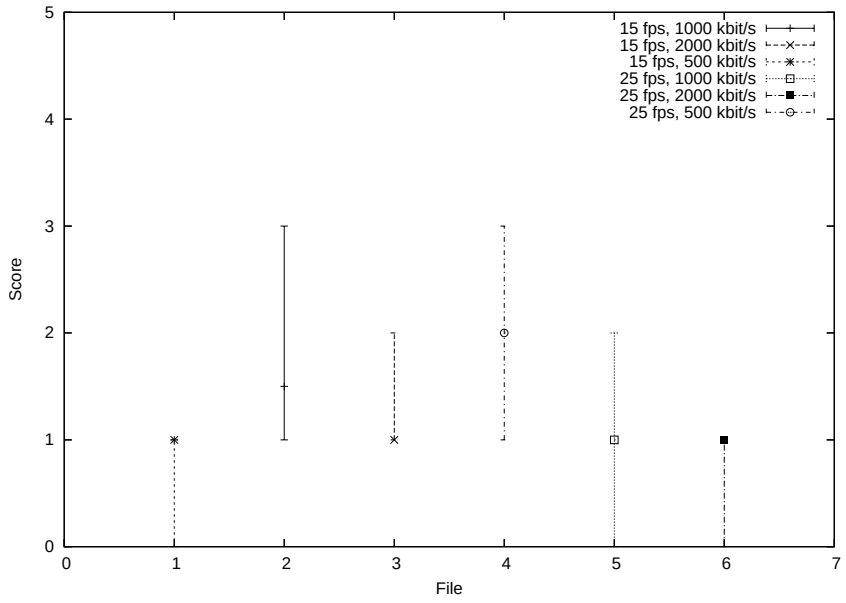

Figure 4.20: High resolution, min/max/median

Some problems caused by a data set that was too small were immediately observable. As the bitrate increased there should have been a rise in quality, i.e. the DSCQS score was lowered since it described the amount of perceived difference between clips. But in the high resolution class graph, the clips with the lowest settings had a very good score. A small number of samples as well as a limited variety of clips might have caused this anomaly. In addition, feedback from the participants suggested that some clip variations were perceived as being very close to the reference, which was also suggested by the objective tests. DSCQS scores would then naturally be low. If the anomaly was caused by outliers then the problem could have been identified and removed by using a larger data set.

Another issue was the similarity between some parameter settings. Apparently the parameter variations should have been more numerous and/or spaced further apart. In the clips *evil* and *princess* this was particularly noticeable, with several parameter combinations and variations getting the same DSCQS score.

4.3 Test Results Discussion

As mentioned before, comparing the objective HVS based metrics with subjective metrics could be beneficial for future experiments. To verify the suitability of an objective metric the results from it must match those produced by one or more subjective metrics. The project could not test this since little to no per clip data was available from the subjective tests. Based on comments from several of the

test participants regarding how they rated the video clips, SSIM and VQM seemed to give similar results. Promising as those metrics were, without proper testing this was speculation.

Several of the graphs showed non-linear quality development. In the objective data set this was easily observed for some of the clips. Consider the graph in fig. B.26. In this graph the difference between 500 kbit/s and 1000 kbit/s was larger than between 1000 kbit/s and 2000 kbit/s, despite the latter having a doubled increase in amount of bits. The same trend could be observed in the other tests where the encodings were not getting bitrate saturated. In theory it should also have been possible to observe the inverse in the very low bitrate section. The curve would rise slowly up to a point where it would go up sharply, at least for the HVS based metrics. This would be caused by the image distortion being so high at very low bitrates that one must get to a certain point before the image content becomes identifiable. However, since the project did not test very low bitrates, this could not be confirmed.

The same type of quality development pattern should also in theory have been possible to observe in the subjective data set, but the severely limited number of samples made this very difficult. The human eye is more sensitive to severe differences between clips, and so smaller differences get progressively more difficult to notice, to a point where there is no discernible difference. At that point the impaired clip is considered being at the same quality level as the reference. This is different from objective results, where the impaired clip will never reach the exact same quality level as the reference due to the nature of lossy video compression. For low bitrate video the quality should rise slowly until the HVS can in some way identify the content to be able to associate and compare it with something. From there the quality rises sharply as more bits at that point will have a more profound effect than later, due to the degree of “missing” data and lack of precision.

4.4 Utility Function

To measure quality development for the various parameter changes a utility function was needed. This function would measure video quality in terms of difference from the reference, for instance in percent. As a starting point because it seemed to fit well with the development description in the previous section, the inverted exponential function was proposed (fig. 4.21).

$$F(x) = b(1 - e^{-ax})$$

Figure 4.21: Basic exponential utility function

Here b was the upper limit which represented maximum quality (same quality as reference). However, some extra requirements were needed. As described in the previous section the quality value may start slowly, proceed to rise sharply, and

then flatten out. The utility function in fig. 4.22, being expanded upon from the previous basic exponential function, had these properties. Had there been more parameter variations in the tests this function could have been verified by using regression to find the best curve fit.

$$F(x) = b(1 - e^{-ax^2+cx})$$

Figure 4.22: Utility function

The approach taken for the thesis to generate a quality metric was to describe quality as a function of the parameters by using conditional functions, which used static parameters to get an exponential function in two dimensions. These functions would then describe a "slice" of the n-dimensional parameter space. A conditional function in this context would be one showing quality development in only two dimensions: quality and one parameter. The selected parameter was the only variable, as the other parameters were static. This meant there was one function for each parameter variation tested. As an example, the framerate functions would have the form seen in fig. 4.23. Bitrate, resolution and any other parameter would get their own functions of this kind.

$$F_{\substack{res=i*j \\ bitrate=k}}(fps) = b(1 - e^{-ax^2+cx})$$

Figure 4.23: Example conditional function

Here resolution and fps were static, and would produce the versions in fig. 4.24 based on the tested parameters for the clip *daggers*. As each of these functions only described one parameter variation at a certain point determined by the static parameters, all of them were needed. Through interpolation the quality score could be found for parameter values not tested directly.

$$
\begin{aligned}
F_{\substack{res=640*272 \\ bitrate=500}}(fps) &= b(1-e^{-ax^2+cx}) \\
F_{\substack{res=640*272 \\ bitrate=1000}}(fps) &= b(1-e^{-ax^2+cx}) \\
F_{\substack{res=640*272 \\ bitrate=2000}}(fps) &= b(1-e^{-ax^2+cx}) \\
F_{\substack{res=784*336 \\ bitrate=500}}(fps) &= b(1-e^{-ax^2+cx}) \\
F_{\substack{res=784*336 \\ bitrate=1000}}(fps) &= b(1-e^{-ax^2+cx}) \\
F_{\substack{res=784*336 \\ bitrate=2000}}(fps) &= b(1-e^{-ax^2+cx})
\end{aligned}
$$

Figure 4.24: Conditional functions for fps parameter of *daggers*

Chapter 5

Conclusion

The project wanted to develop a method for evaluating subjective quality of scalable video. Scalable video was defined as having continuous parameter changes. None of the current metrics, neither objective nor subjective, were designed with this in mind.

To achieve this goal the project had to define a way to produce appropriate test material. A method was designed to both extract reference video and generate impaired video sequences. Instructions for how to set up a streaming video environment and produce video files with network metric based impairments was also made (but not used). Further, a method was designed to evaluate video objectively and subjectively. For the subjective tests custom software was made to perform DSCQS testing adapted to the needs of the project. Finally, the results from the tests were analyzed and discussed. Comparisons were made between the objective and subjective results in order to find common ground between them. That comparison was severely limited by the inaccuracies of the subjective results. However, most of the objective test scores could be considered to lie in a range which should equal a fairly good image quality. The DSCQS scores told the same story as both the scores and feedback from the participants indicated it was generally difficult to see major differences between the compared clips. This insinuated that the HVS based objective tests could, at least to some degree, be used in place of subjective testing.

A proposal was made for a utility function that described how subjectively perceived quality developed as the parameters changed. If implemented, it would enable tracking of quality changes in real time as the parameters were altered by the streaming server. The proposed utility function was not verified due to lack of time and having a test result set that was too small. It became apparent during testing that the choice of parameter variations were not spaced far enough apart, giving inaccurate results regarding quality development for a parameter. There were also too few parameter variations. In the case of fps and resolution there were only two, which was not enough to make a curve estimation. Despite this some general trends could be observed. Even though the merged data set for the subjective tests had low

statistical accuracy, the lack of high DSCQS scores showed that the participants generally tolerated the quality differences fairly well. Because of the data and parameter limitations, it was unclear if this was caused by a high tolerance to the impairments or by not having dramatic enough parameter variations.

The subjective test was not complete enough to be fully usable. Further testing would require an experiment on a larger scale. More test participants were needed, and each had to watch a larger set of test clips with more parameter variations in order to get high enough statistical resolution. Some testing needed to be done to find upper and lower boundaries for the parameters so they could be suitably spaced apart. Also, more tests should be run to find how well the HVS based objective test results (i.e., from SSIM and VQM) correlate with subjective results. Using such objective tests would enable testing with a much larger set of parameter variations since they would require less time and organization than subjective test methods.

Bibliography

[1] JPEG Committee. The JPEG Image Compression Standard. *ISO/IEC IS 10918-1*.

[2] Microsoft Corporation. Microsoft. http://www.microsoft.com/.

[3] Brian P. Crow, Indra Widjaja, Jeong Geun Kim, and Prescott T. Sakai. IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine*, September 1997.

[4] Milan Cutka. ffdshow codec library. http://sourceforge.net/projects/ffdshow/.

[5] Doom9. Doom9 Video Compression Forum. http://doom9.net/.

[6] Robin Dunn. wxpython: wxwidgets bindings for python. http://wxpython.org/.

[7] Ericsson. EDGE - Introduction of high-speed data in GSM/GPRS networks. http://www.ericsson.com/technology/whitepapers/edge_wp_technical.pdf.

[8] Jean Le Feuvre. GPAC. http://gpac.sourceforge.net/.

[9] gabest and more. Media Player Classic. http://sourceforge.net/projects/guliverkli/.

[10] Donald Graft. DGMPEGDec. http://neuron2.net/dgmpgdec/dgmpgdec.html.

[11] Stephen Hemminger. iproute2. http://linux-net.osdl.org/index.php/Iproute2.

[12] Stephen Hemminger. Linux Network Emulation. http://linux-net.osdl.org/index.php/Netem.

[13] Apple Computer Inc. Darwin Streaming Server. http://developer.apple.com/darwin/projects/streaming/.

[14] DivX Inc. Divx Video Codec. http://www.divx.com/.

[15] Live Networks Inc. Live555 Streaming Media. http://www.live555.com/.

[16] ITU-R. Methodology for the subjective assessment of the quality of television pictures. *Recommendation ITU-R BT.500-11*, 2002.

[17] Cheng Jin, David X. Wei, and Steven H. Low. FAST TCP. In *Proceedings of IEEE Infocom, Hong Kong*, March 2004.

[18] Charles Krasic. *A Framework for Quality-Adaptive Media Streaming: Encode Once - Stream Anywhere*. PhD thesis, Oregon Health & Science University, OGI School of Science & Engineering, February 2004. `http://www.cs.ubc.ca/~krasic/`.

[19] Christoph Lampert, Pascal Massimino, Michael Militzer, Peter Ross, and Various. XviD MPEG-4 Video Codec. http://www.xvid.org/.

[20] Avery Lee. VirtualDub. http://www.virtualdub.org/.

[21] Weiping Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3), March 2001.

[22] David Mackie, Bill May, and Alix M. Franquet. MPEG4IP. http://mpeg4ip.sourceforge.net/.

[23] Mike Matsnev. Haali Media Splitter. http://haali.cs.msu.ru/mkv/.

[24] Moving Picture Experts Group (MPEG). The MPEG-2 Standard. *ISO/IEC 13818-5:2005*.

[25] M. Pinson and S. Wolf. Comparing subjective video quality testing methodologies. Institute for Telecommunication Sciences (ITS), National Telecommunications and Information Administration (NTIA), U.S. Department of Commerce.

[26] Petri Possi. UMTS World. http://www.umtsworld.com/.

[27] Raj Kumar Rajendran, Mihaela van der Schaar, and Shih-Fu Chang. FGS+: Optimizing the Joint SNR-Temporal Video Quality in MPEG-4 Fine Grained Scalable Coding.

[28] RealNetworks. SureStream. http://service.real.com/help/library/guides/ProductionGuide/prodguide/htmfiles/realsys.htm#64854.

[29] Ben Rudiak-Gould, Edwin van Eggelen, and more. AviSynth. http://avisynth.org/.

[30] VQEG Study Group 9. Final report from the Video Quality Experts Group on the validation of objective models of video quality assessment. *Rapporteur Q11/12 (VQEG)*, June 2000.

[31] MPlayer Team. MEncoder. http://www.mplayerhq.hu/.

[32] Mihaela van der Schaar and Hayder Radha. A Hybrid Temporal-SNR Fine-Granular Scalability for Internet Video. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3), March 2001.

[33] Guido van Rossum, Python Software Foundation, and more. The Python Programming Language. http://python.org/.

[34] Dmitriy Vatolin, Alexey Moskvin, Oleg Petrov, and Nicolay Trunichkin. MSU Video Quality Measurement Tool. http://www.compression.ru/index_en.htm.

[35] Dmitriy Vatolin and Oleg Petrov. MSU Perceptual Video Quality Tool. http://www.compression.ru/index_en.htm.

[36] Visumalchemia.com. Video Quality Studio. http://www.visumalchemia.com/.

[37] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13, April 2004.

[38] A.B. Watson. Toward a perceptual video quality metric. *Human Vision, Visual Processing, and Digital Display VIII*, 1998.

[39] Thomas Williams, Colin Kelley, Russell Lang, Dave Kotz, John Campbell, Gershon Elber, Alexander Woo, and more. gnuplot. http://gnuplot.info/.

[40] Feng Xiao. DCT-based Video Quality Evaluation. http://www-ise.stanford.edu/class/ee392j/projects/projects/, 2000.

Appendix A

GnuPlot Scripts

```
set xlabel ‘‘Frame’’
set ylabel ‘‘PSNR’’
set xrange [0:250]
set yrange [28.0:46.0]
set data style lines
set output ‘‘outfile.ps’’
set terminal postscript eps
plot ‘‘datfile_br500.dat’’ title ’500 kbit/s’, \
‘‘datfile_br1000.dat’’ title ’1000 kbit/s’, \
‘‘datfile_br2000.dat’’ title ’2000 kbit/s’
set output ‘‘outfile.png’’
set terminal png
plot ‘‘datfile_br500.dat’’ title ’500 kbit/s’, \
‘‘datfile_br1000.dat’’ title ’1000 kbit/s’, \
‘‘datfile_br2000.dat’’ title ’2000 kbit/s’
```

Figure A.1: Line graph example, frame number and test score

```
set xlabel ‘‘File’’
set ylabel ‘‘PSNR’’
set xrange [0:7]
set yrange [28.3466:45.1496]
set output ‘‘outfile_med.ps’’
set terminal postscript eps
plot ‘‘datfile_br1000_med.dat’’ \
title ’15 fps, 1000 kbit/s’ with errorbars, \
‘‘datfile_br2000_med.dat’’ \
title ’15 fps, 2000 kbit/s’ with errorbars, \
‘‘datfile_br500_med.dat’’ \
title ’15 fps, 500 kbit/s’ with errorbars, \
‘‘datfile_br1000_med.dat’’ \
title ’25 fps, 1000 kbit/s’ with errorbars, \
‘‘datfile_br2000_med.dat’’ \
title ’25 fps, 2000 kbit/s’ with errorbars, \
‘‘datfile_br500_med.dat’’ \
title ’25 fps, 500 kbit/s’ with errorbars
set output ‘‘outfile_med.png’’
set terminal png
plot ‘‘datfile_br1000_med.dat’’ \
title ’15 fps, 1000 kbit/s’ with errorbars, \
‘‘datfile_br2000_med.dat’’ \
title ’15 fps, 2000 kbit/s’ with errorbars, \
‘‘datfile_br500_med.dat’’ \
title ’15 fps, 500 kbit/s’ with errorbars, \
‘‘datfile_br1000_med.dat’’ \
title ’25 fps, 1000 kbit/s’ with errorbars, \
‘‘datfile_br2000_med.dat’’ \
title ’25 fps, 2000 kbit/s’ with errorbars, \
‘‘datfile_br500_med.dat’’ \
title ’25 fps, 500 kbit/s’ with errorbars
```

Figure A.2: Error-bar graph example

Appendix B

Objective Test Figures

B.1 PSNR

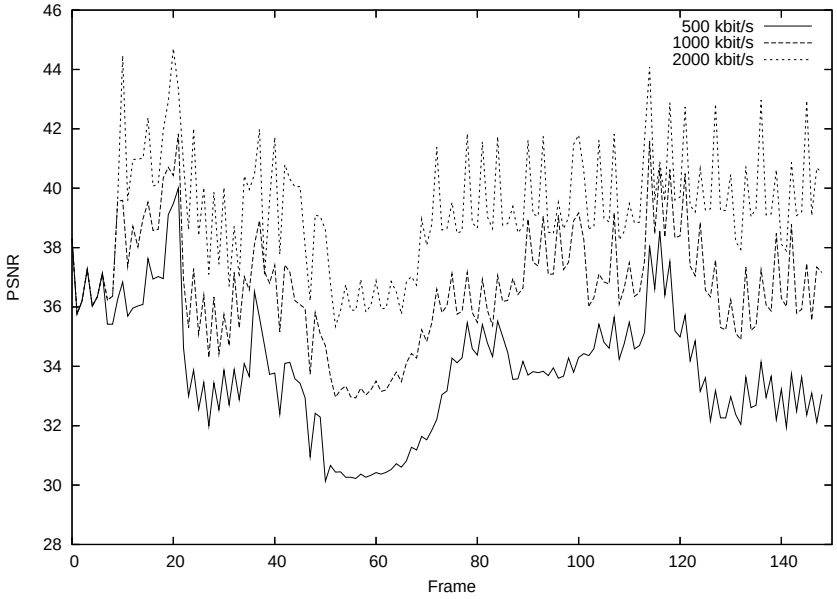

Figure B.1: PSNR, daggers, 640x272, 15 fps

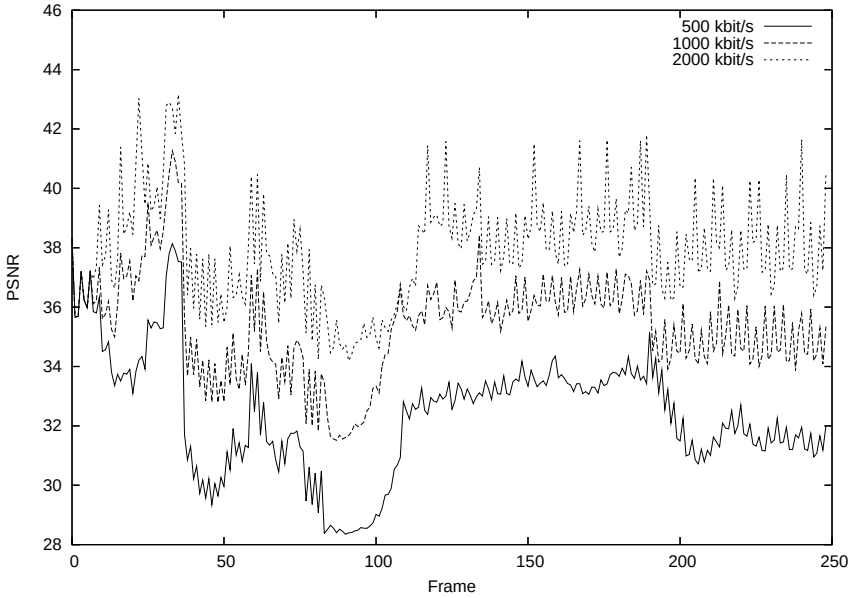

Figure B.2: PSNR, daggers, 640x272, 25 fps

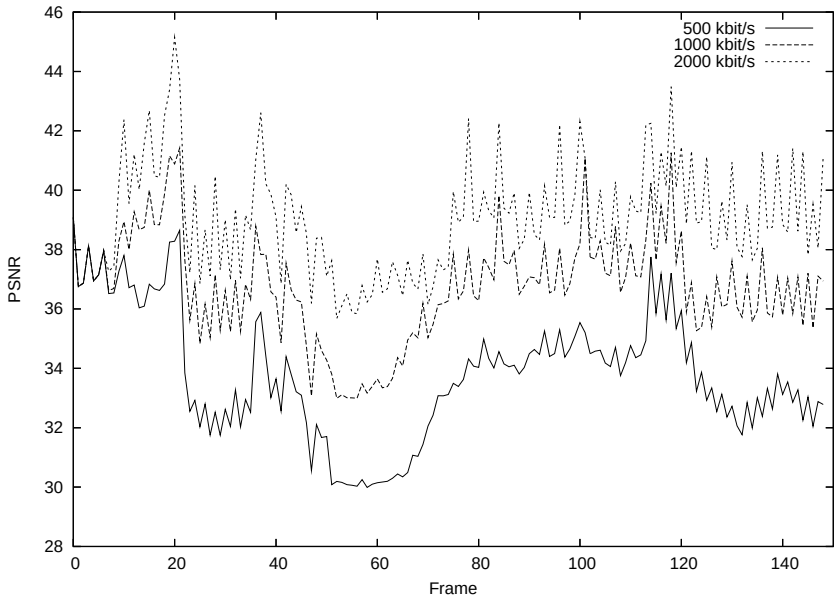

Figure B.3: PSNR, daggers, 784x336, 15 fps

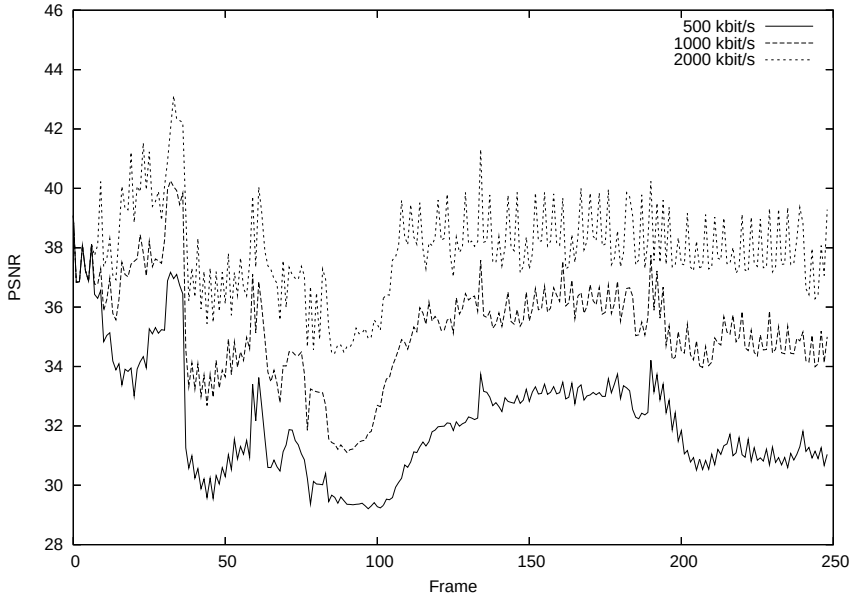

Figure B.4: PSNR, daggers, 784x336, 25 fps

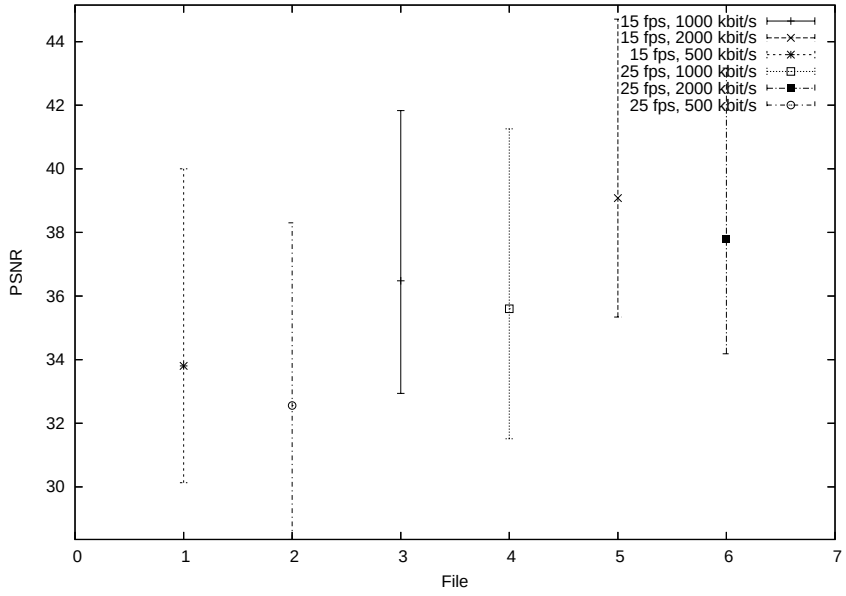

Figure B.5: PSNR, daggers, 640x272, min/max/median

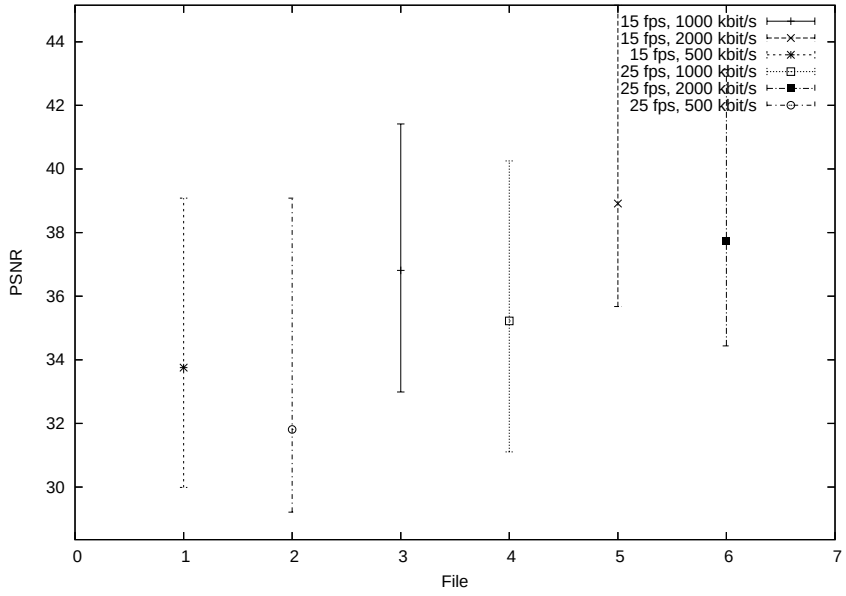

Figure B.6: PSNR, daggers, 784x336, min/max/median

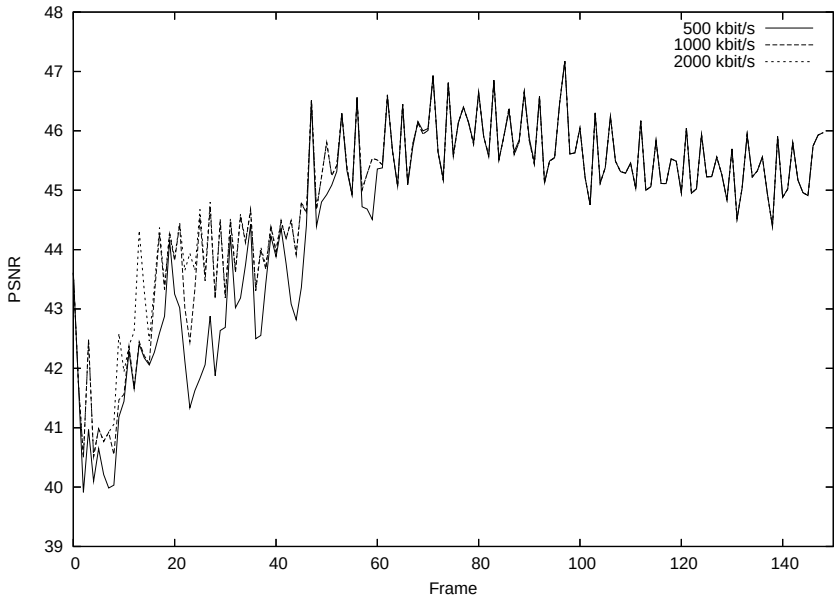

Figure B.7: PSNR, evil, 592x320, 15 fps

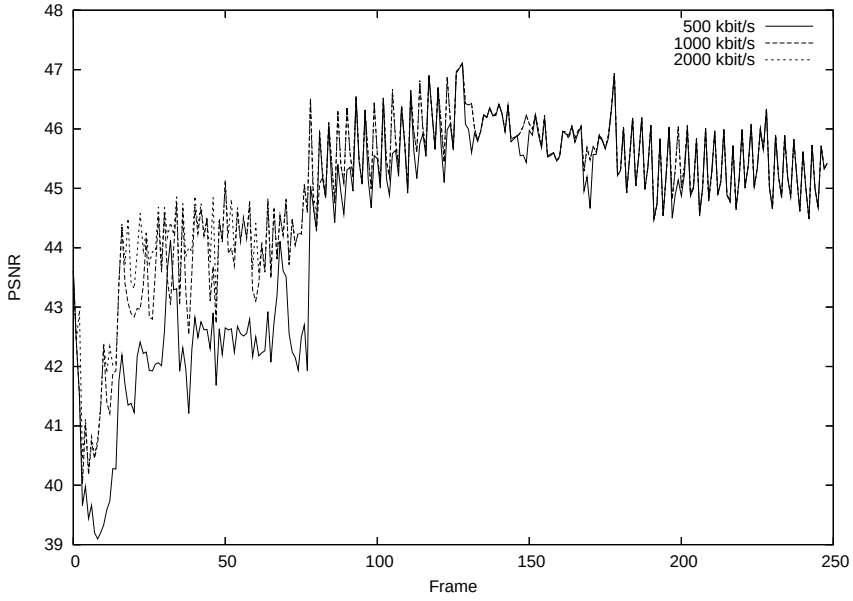

Figure B.8: PSNR, evil, 592x320, 25 fps

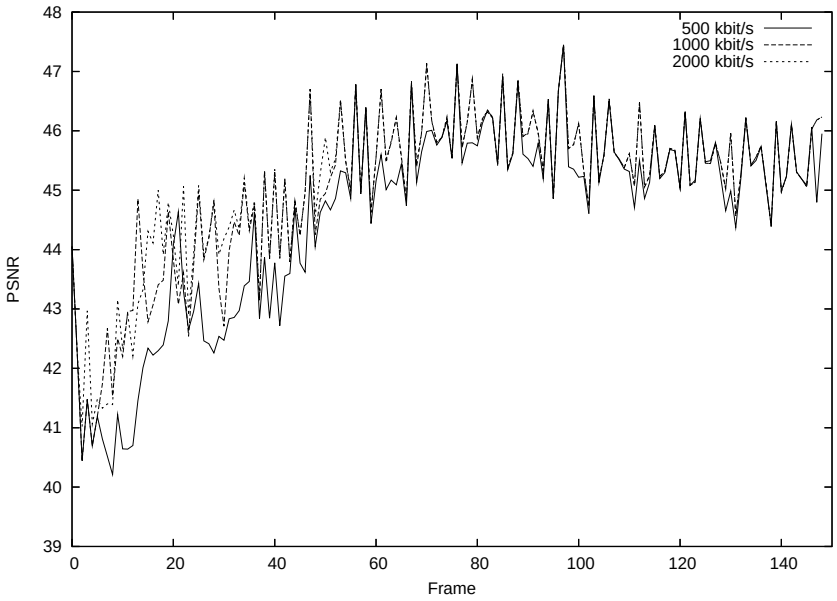

Figure B.9: PSNR, evil, 704x384, 15 fps

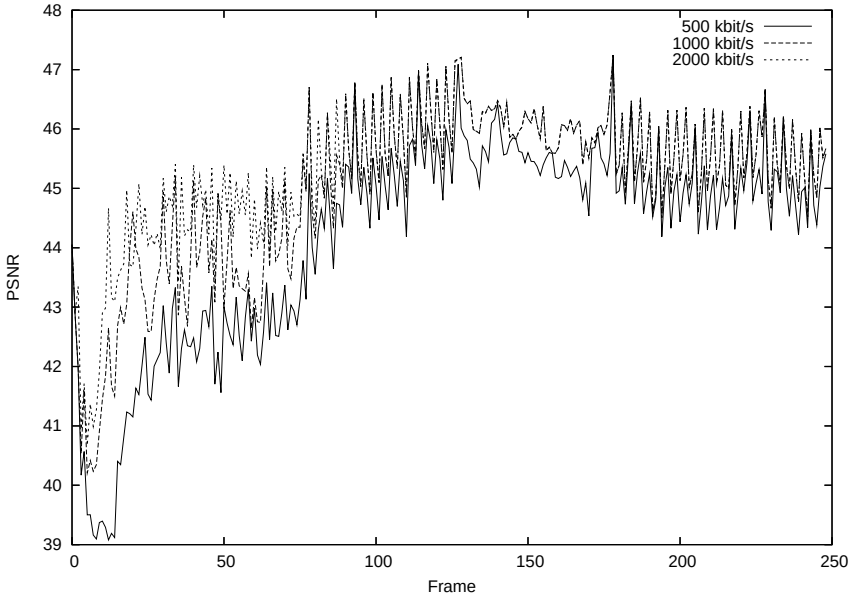

Figure B.10: PSNR, evil, 704x384, 25 fps

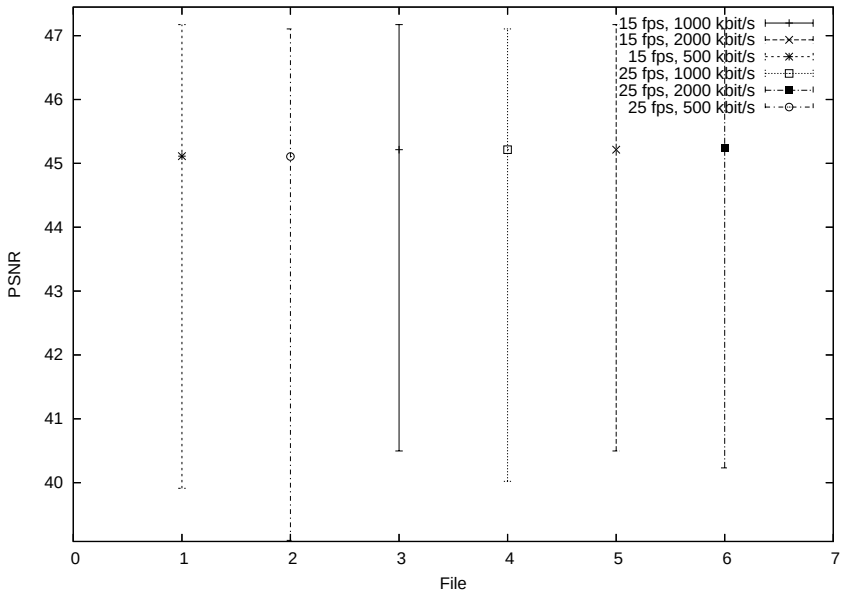

Figure B.11: PSNR, evil, 592x320, min/max/median

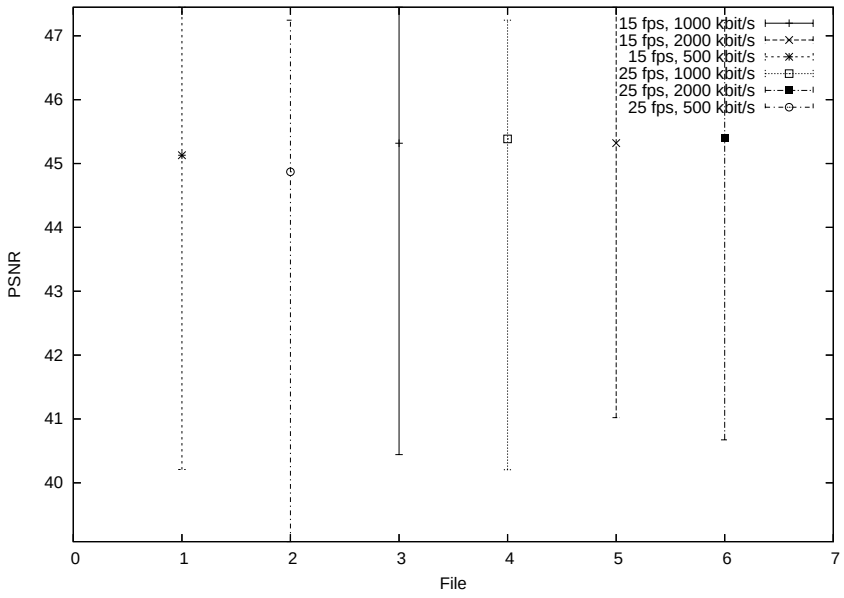

Figure B.12: PSNR, evil, 704x384, min/max/median

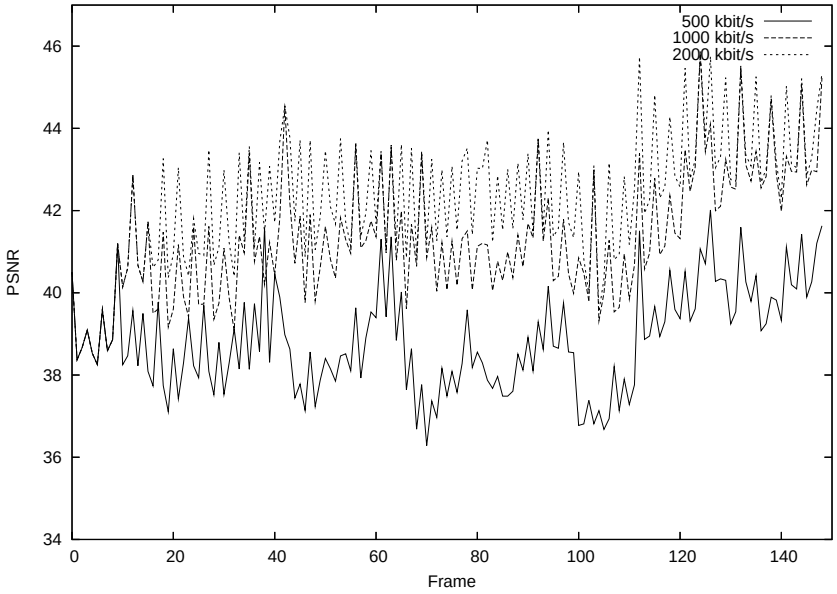

Figure B.13: PSNR, gladiator, 640x272, 15 fps

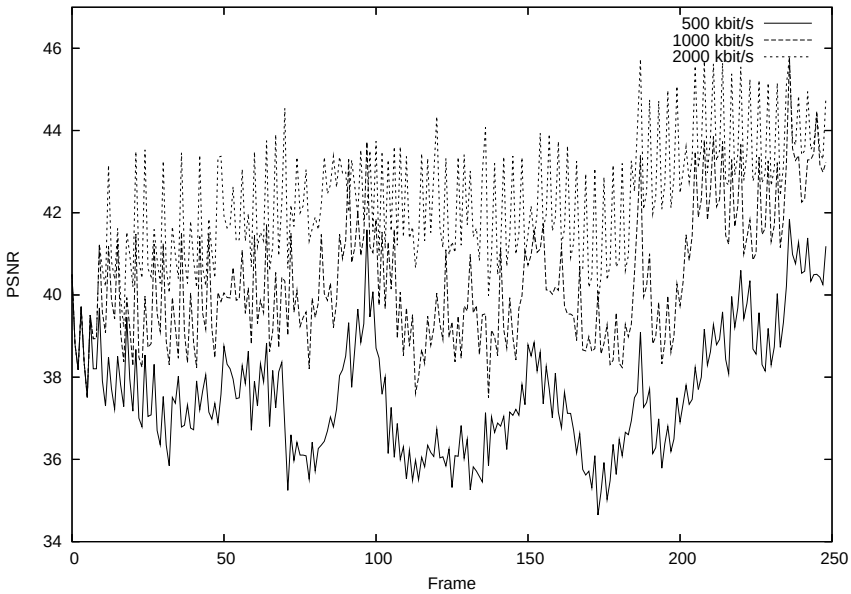

Figure B.14: PSNR, gladiator, 640x272, 25 fps

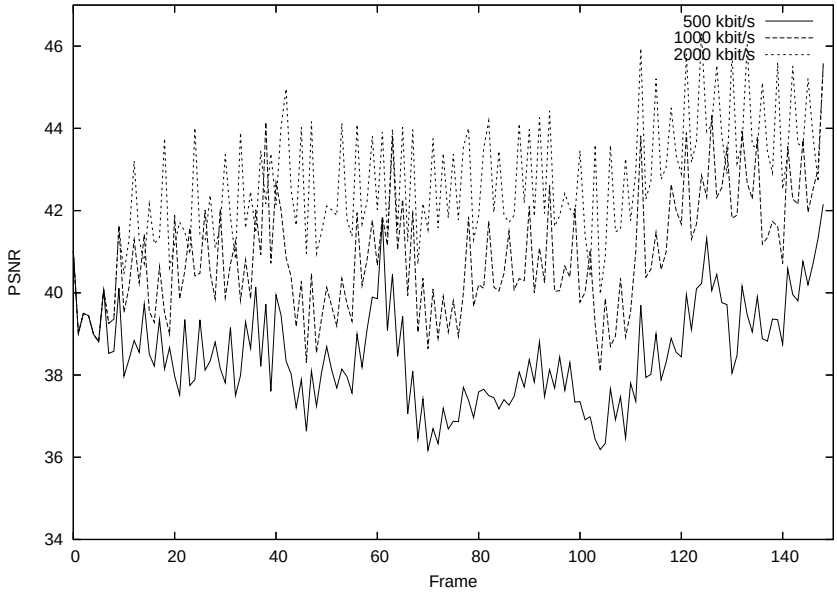

Figure B.15: PSNR, gladiator, 784x336, 15 fps

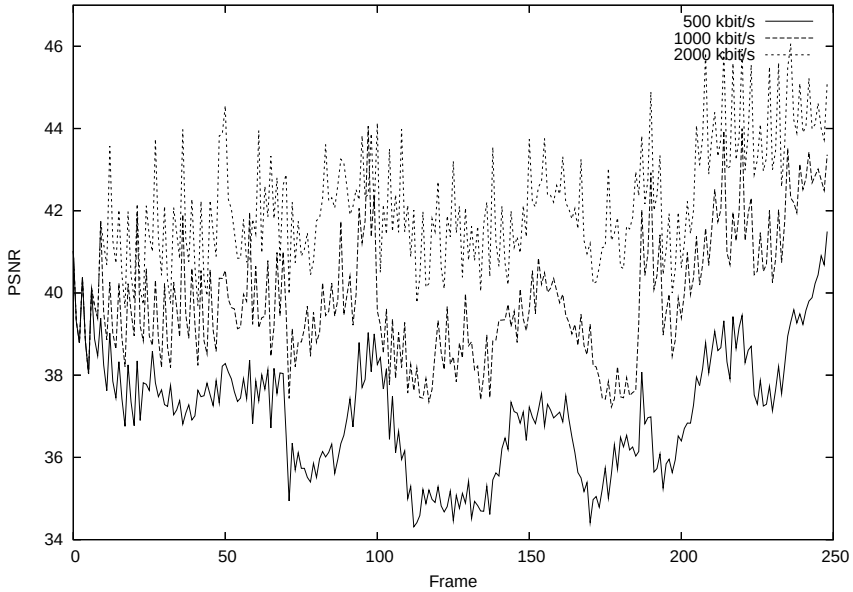

Figure B.16: PSNR, gladiator, 784x336, 25 fps

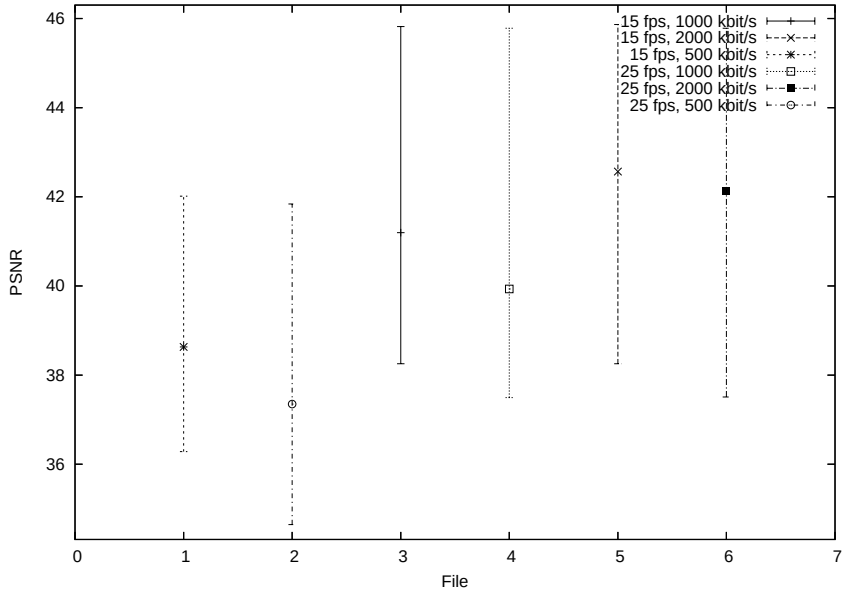

Figure B.17: PSNR, gladiator, 640x272, min/max/median

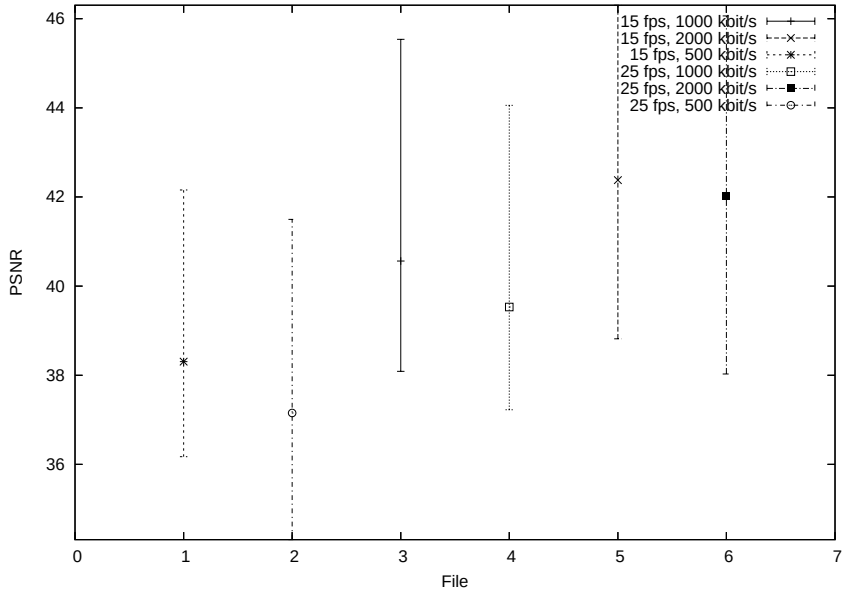

Figure B.18: PSNR, gladiator, 784x336, min/max/median

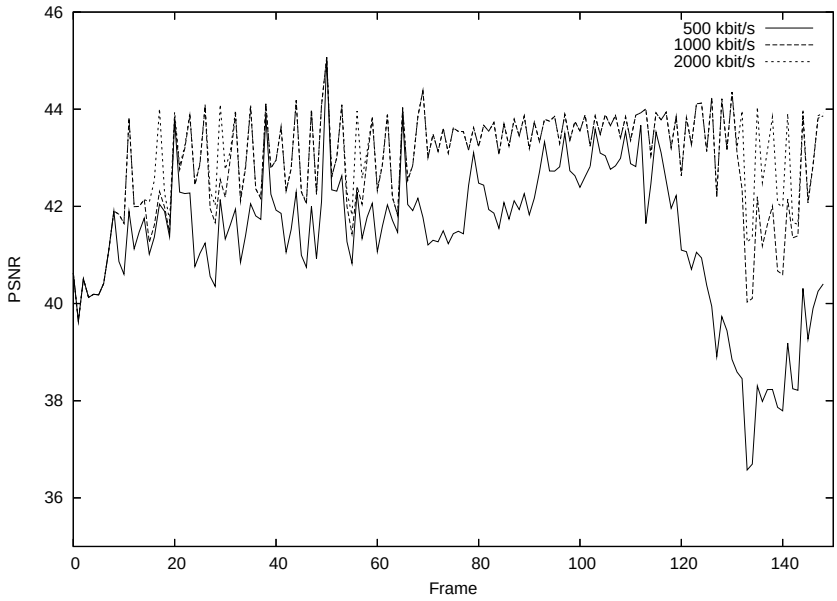

Figure B.19: PSNR, princess, 592x320, 15 fps

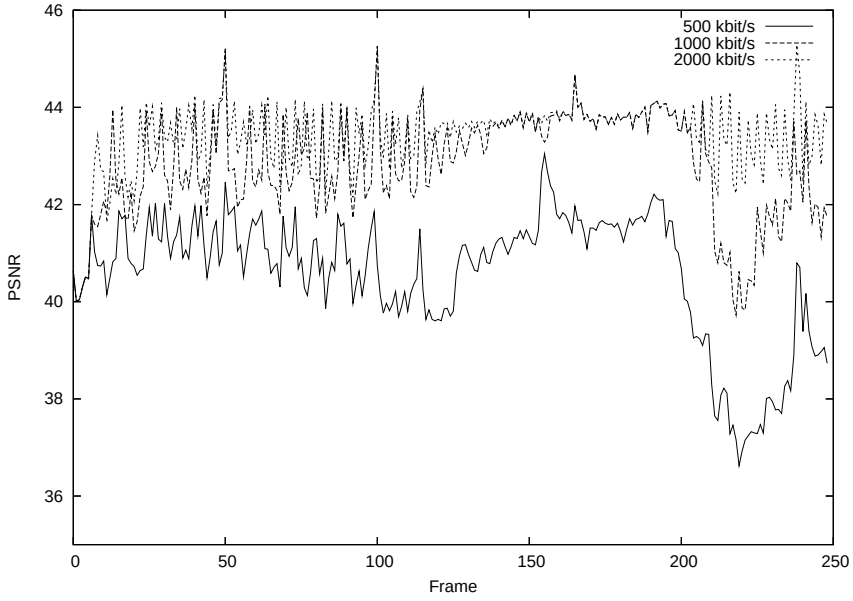

Figure B.20: PSNR, princess, 592x320, 25 fps

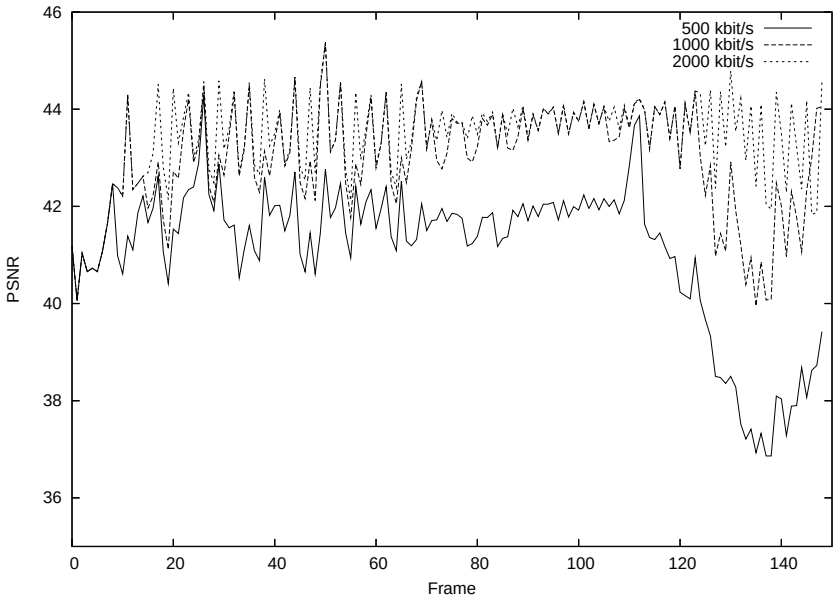

Figure B.21: PSNR, princess, 704x384, 15 fps

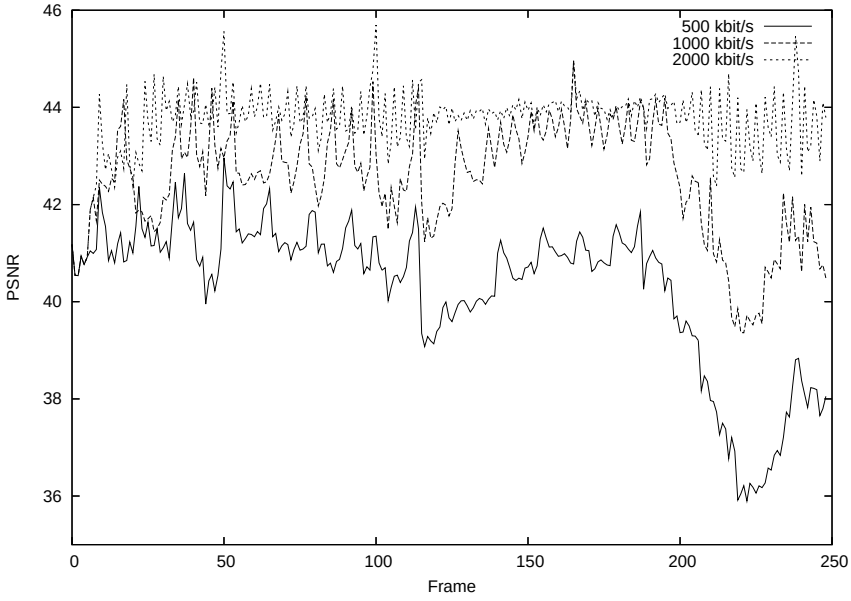

Figure B.22: PSNR, princess, 704x384, 25 fps

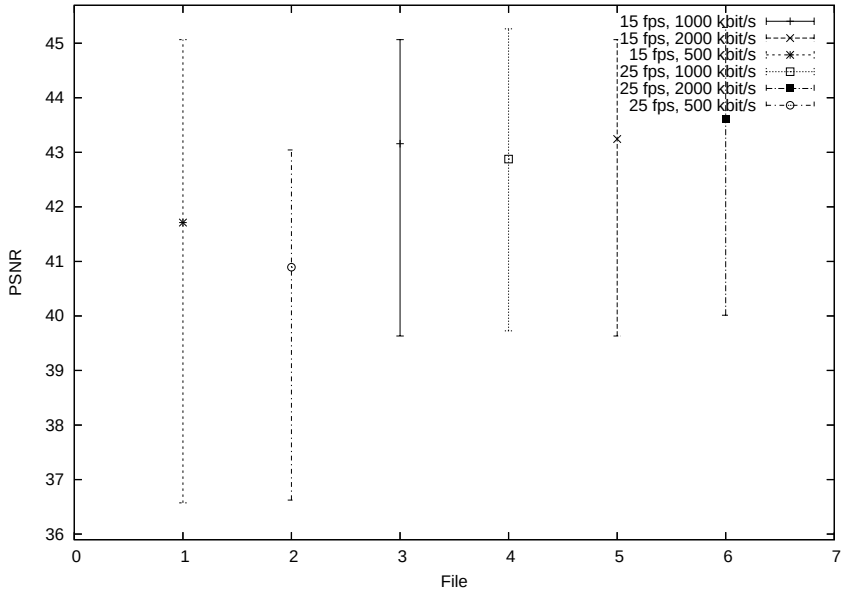

Figure B.23: PSNR, princess, 592x320, min/max/median

B.2 SSIM

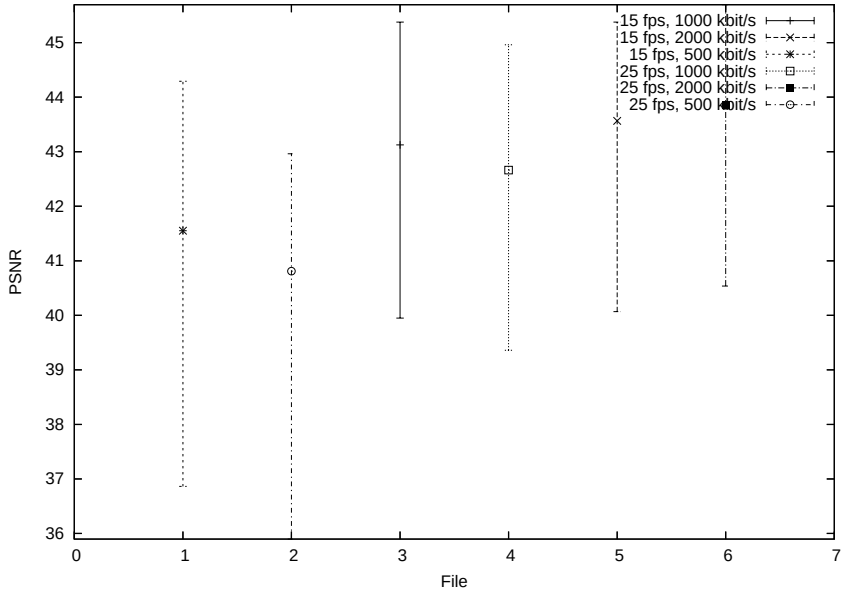

Figure B.24: PSNR, princess, 704x384, min/max/median

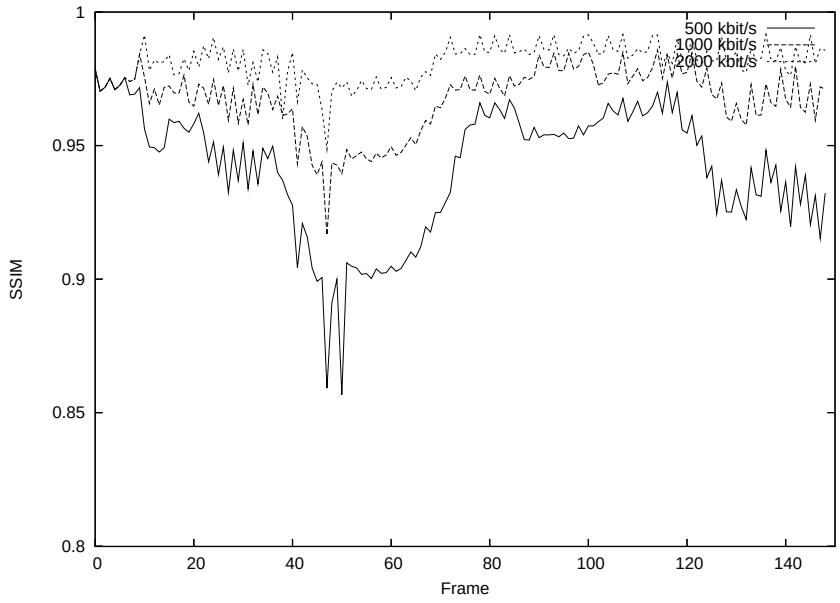

Figure B.25: SSIM, daggers, 640x272, 15 fps

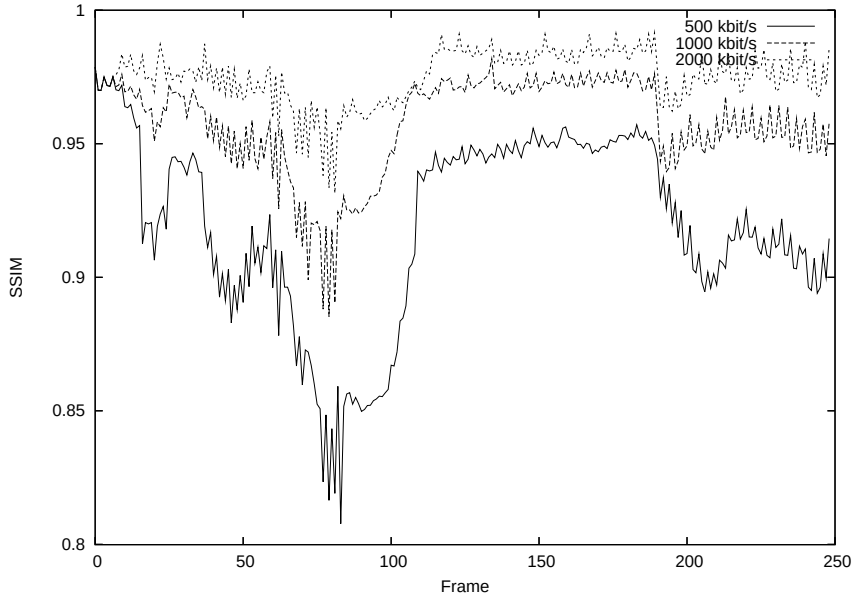

Figure B.26: SSIM, daggers, 640x272, 25 fps

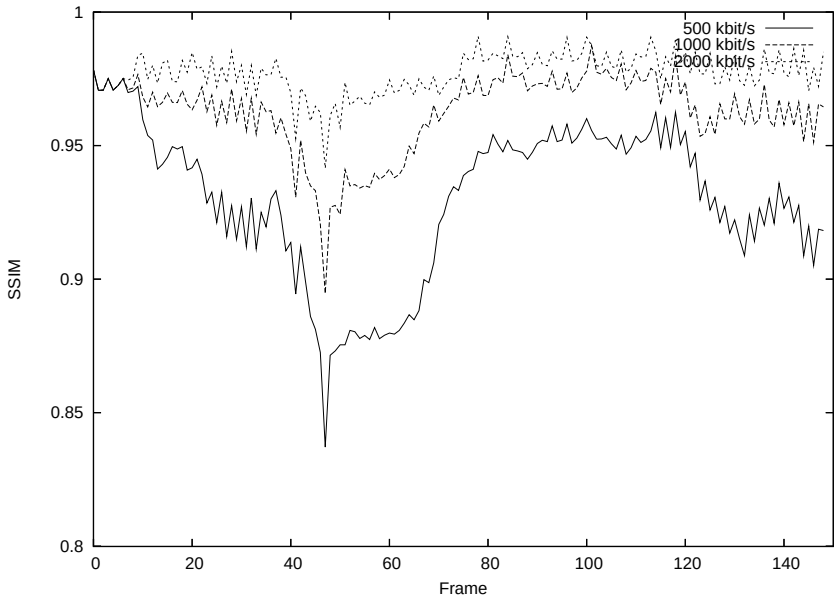

Figure B.27: SSIM, daggers, 784x336, 15 fps

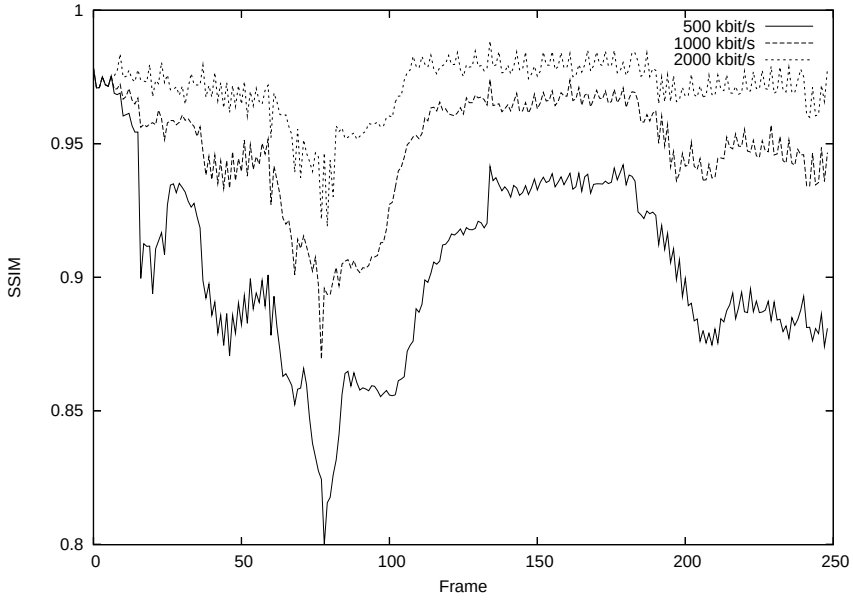

Figure B.28: SSIM, daggers, 784x336, 25 fps

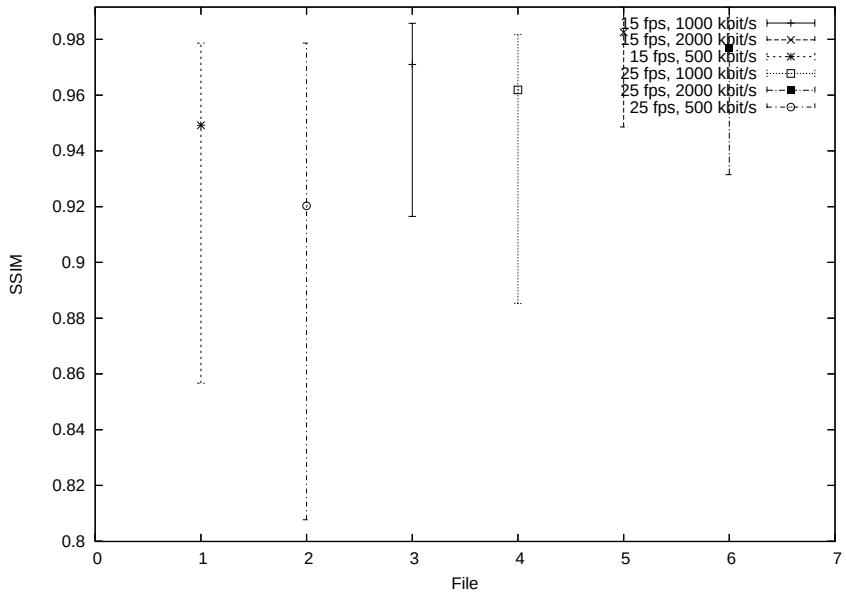

Figure B.29: SSIM, daggers, 640x272, min/max/median

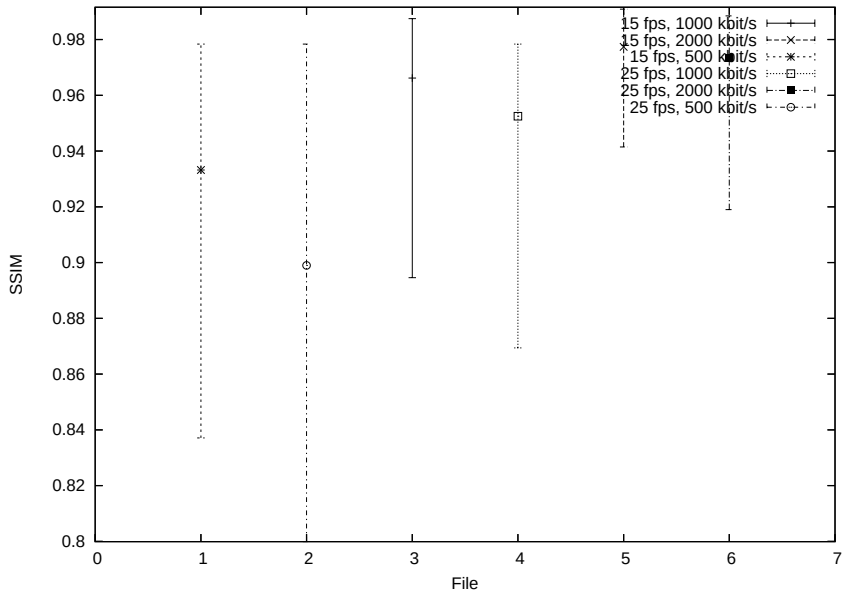

Figure B.30: SSIM, daggers, 784x336, min/max/median

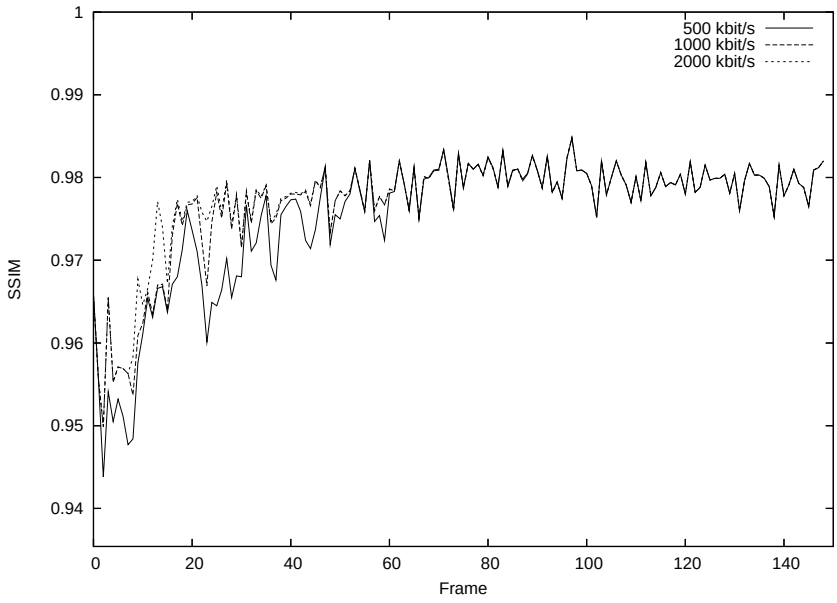

Figure B.31: SSIM, evil, 592x320, 15 fps

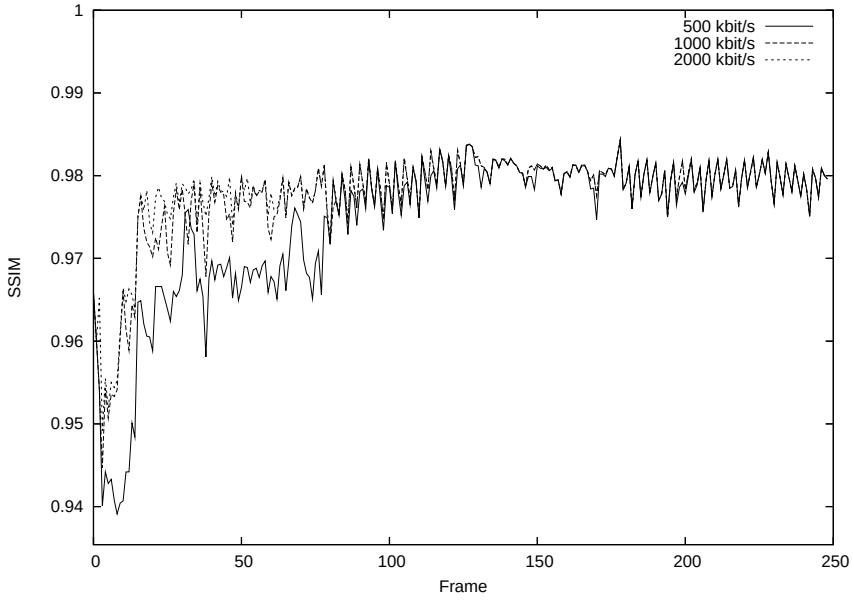

Figure B.32: SSIM, evil, 592x320, 25 fps

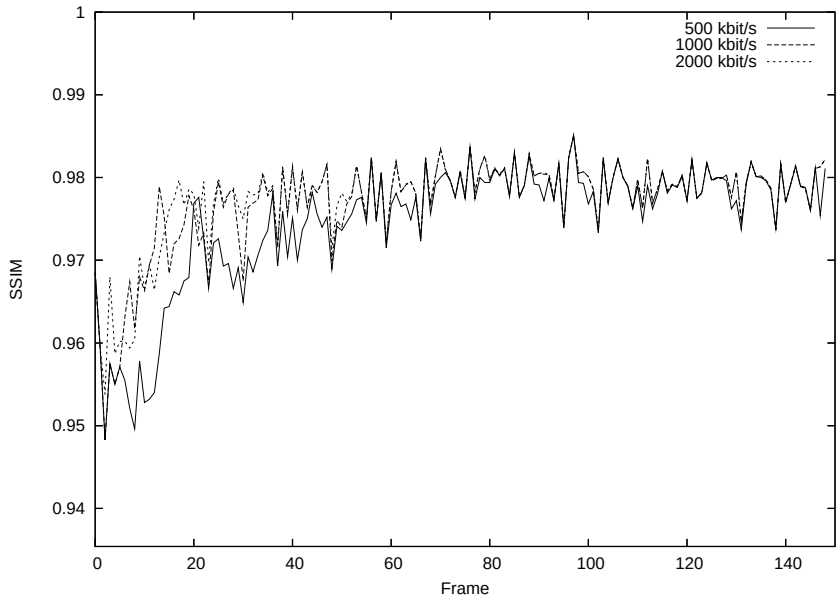

Figure B.33: SSIM, evil, 704x384, 15 fps

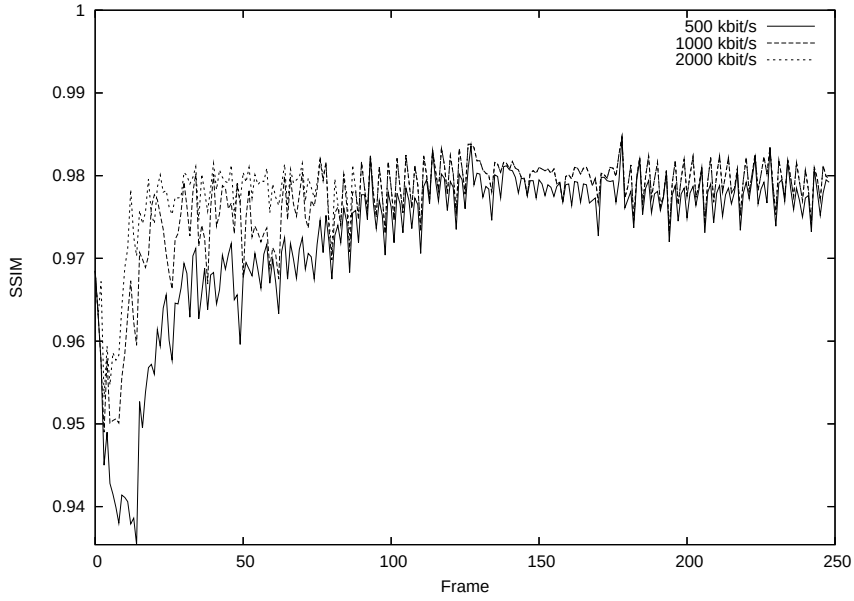

Figure B.34: SSIM, evil, 704x384, 25 fps

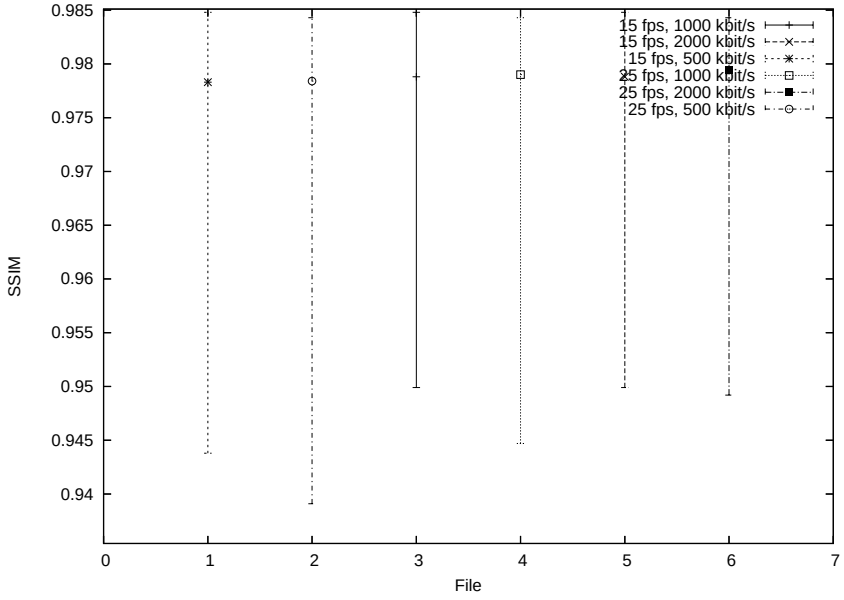

Figure B.35: SSIM, evil, 592x320, min/max/median

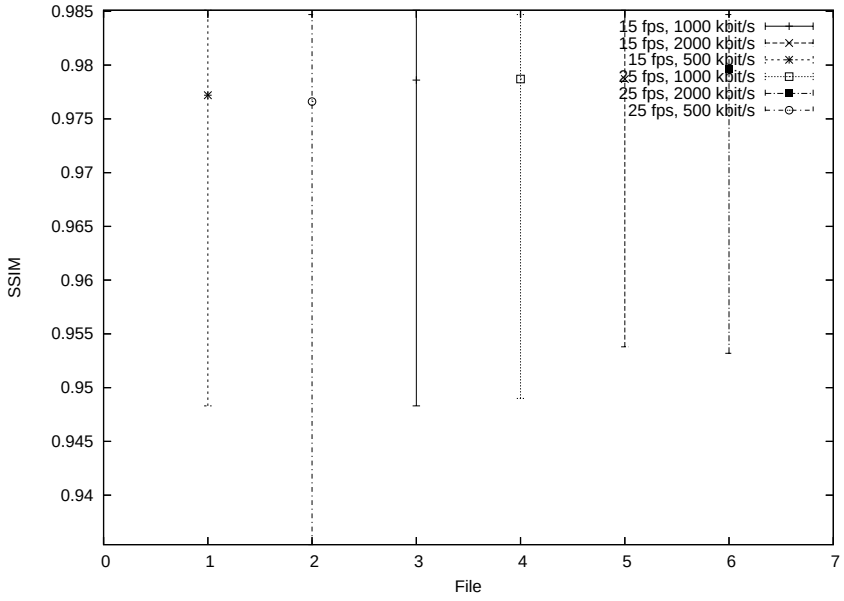

Figure B.36: SSIM, evil, 704x384, min/max/median

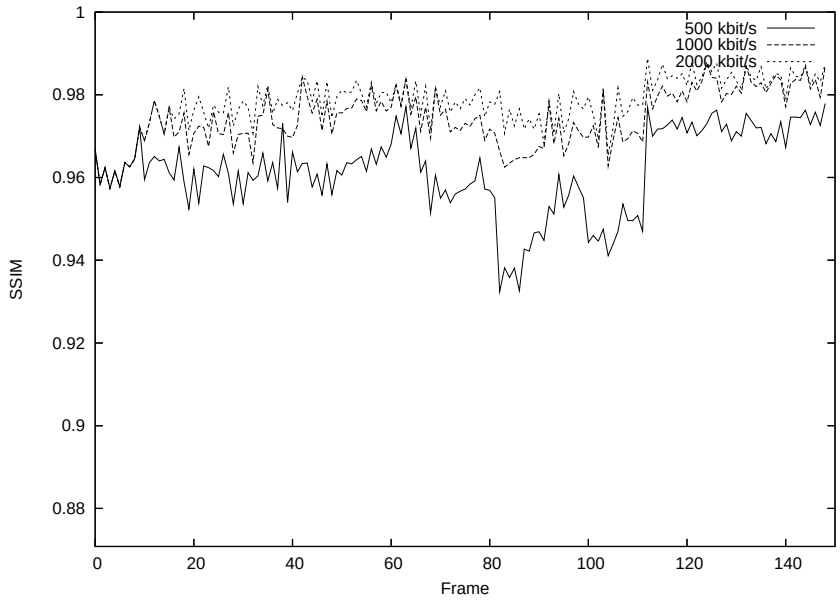

Figure B.37: SSIM, gladiator, 640x272, 15 fps

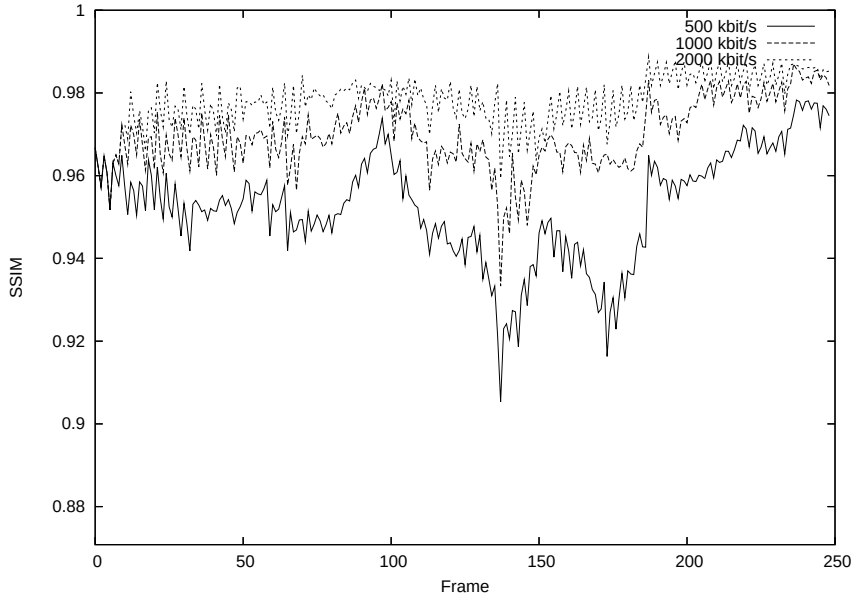

Figure B.38: SSIM, gladiator, 640x272, 25 fps

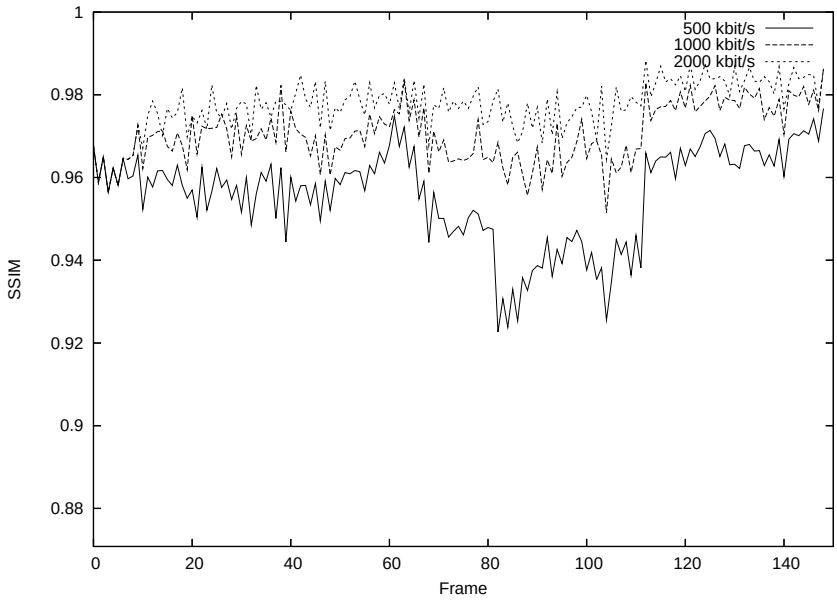

Figure B.39: SSIM, gladiator, 784x336, 15 fps

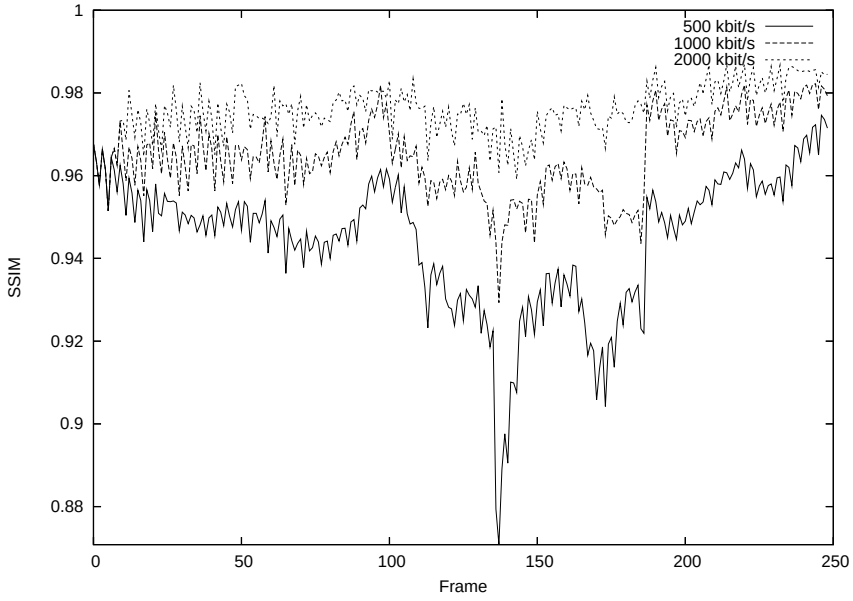

Figure B.40: SSIM, gladiator, 784x336, 25 fps

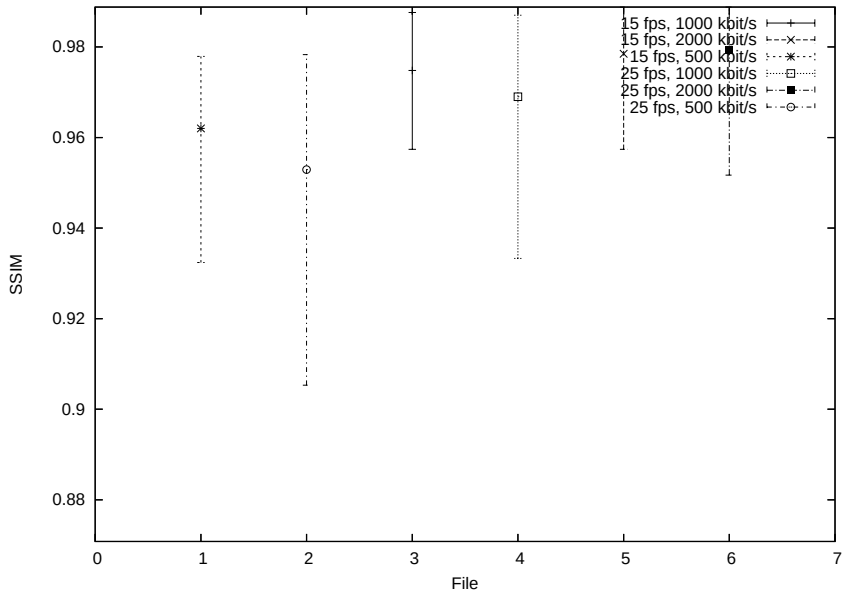

Figure B.41: SSIM, gladiator, 640x272, min/max/median

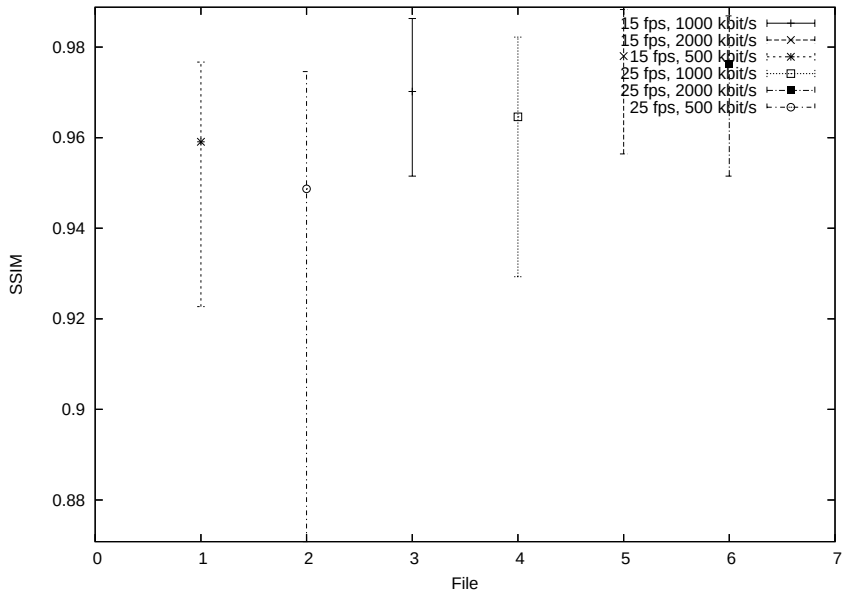

Figure B.42: SSIM, gladiator, 784x336, min/max/median

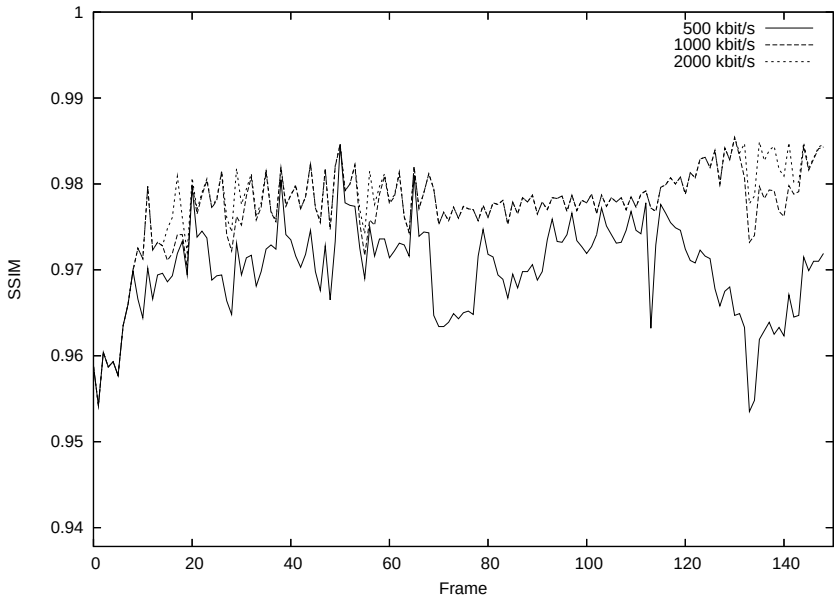

Figure B.43: SSIM, princess, 592x320, 15 fps

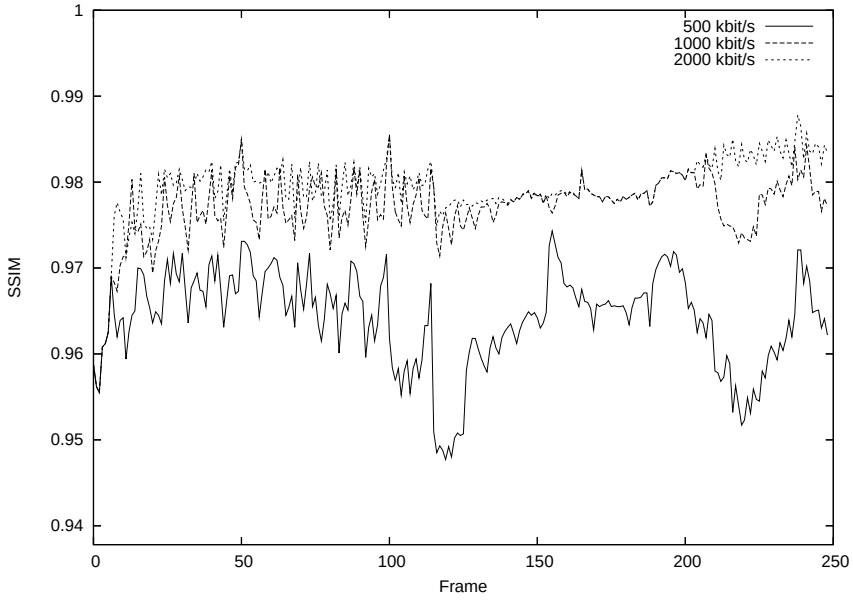

Figure B.44: SSIM, princess, 592x320, 25 fps

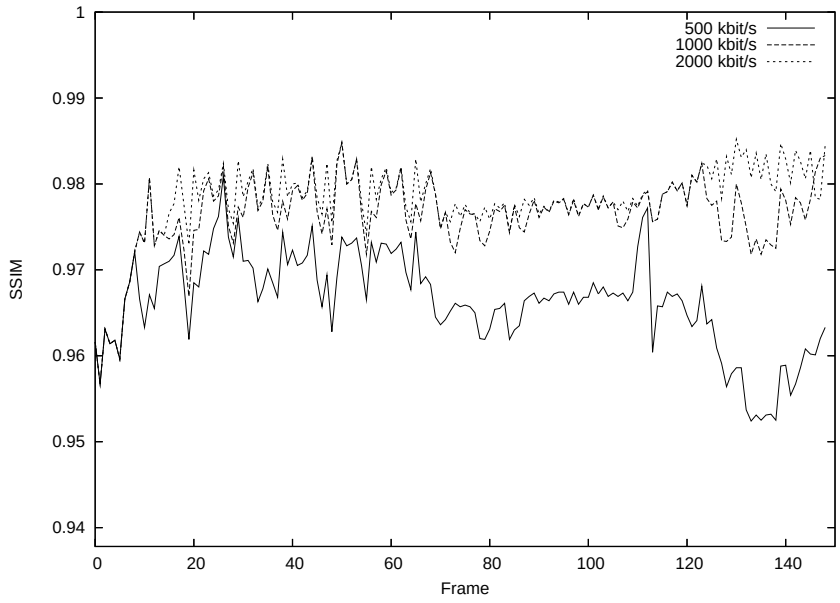

Figure B.45: SSIM, princess, 704x384, 15 fps

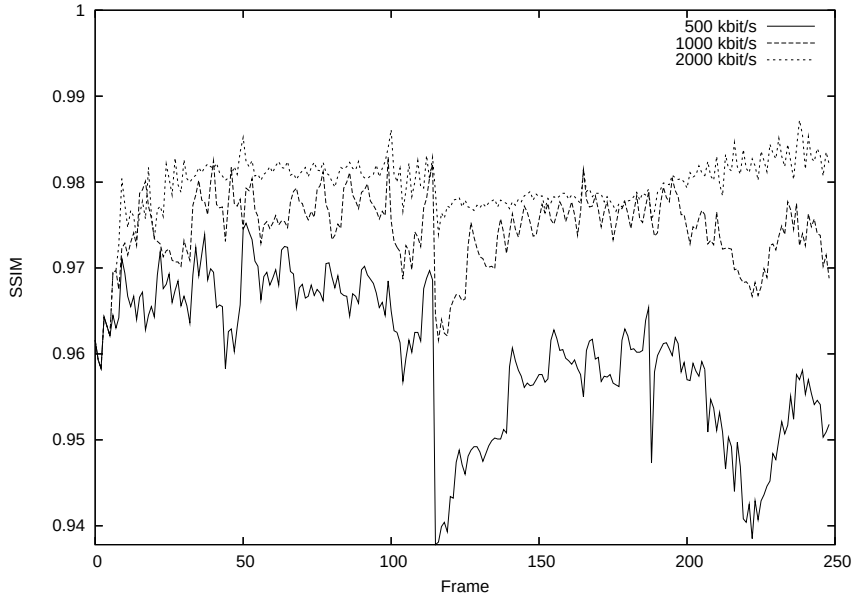

Figure B.46: SSIM, princess, 704x384, 25 fps

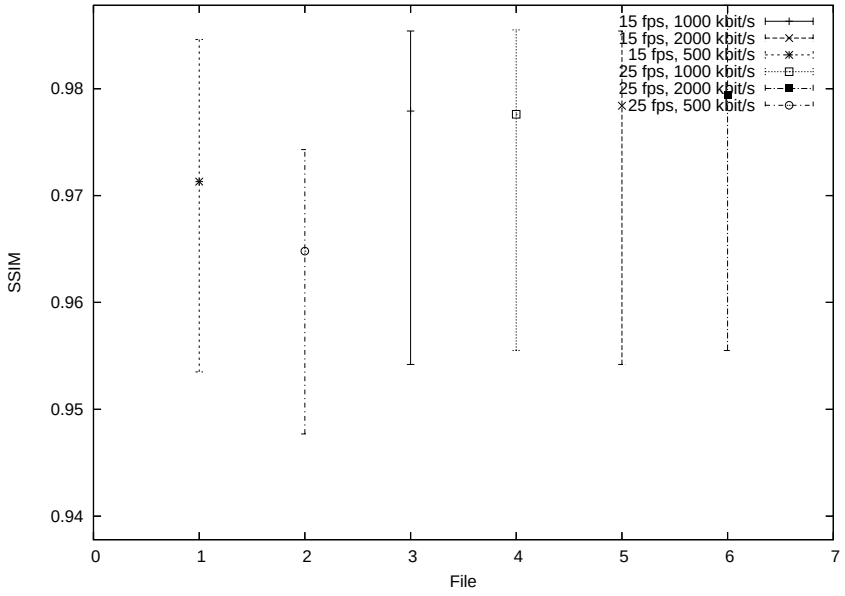

Figure B.47: SSIM, princess, 592x320, min/max/median

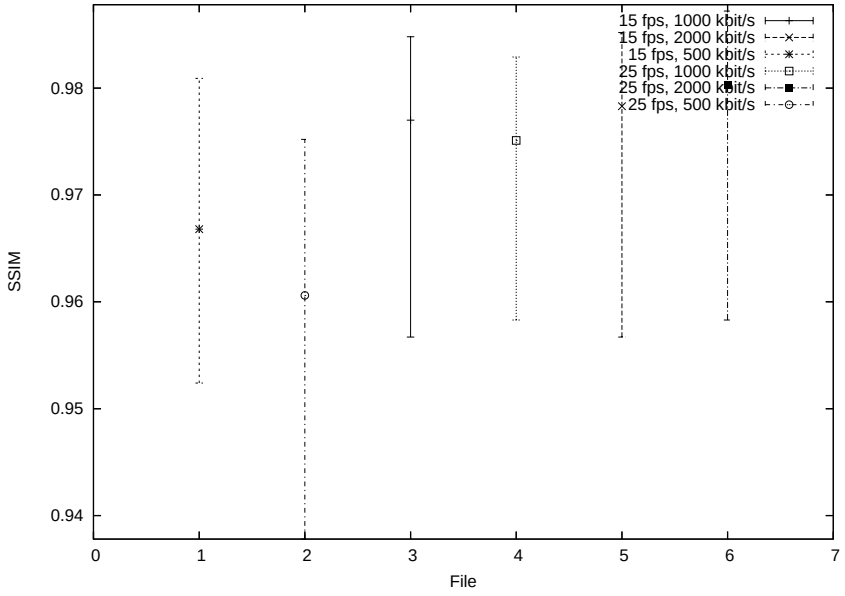

Figure B.48: SSIM, princess, 704x384, min/max/median

B.3 VQM

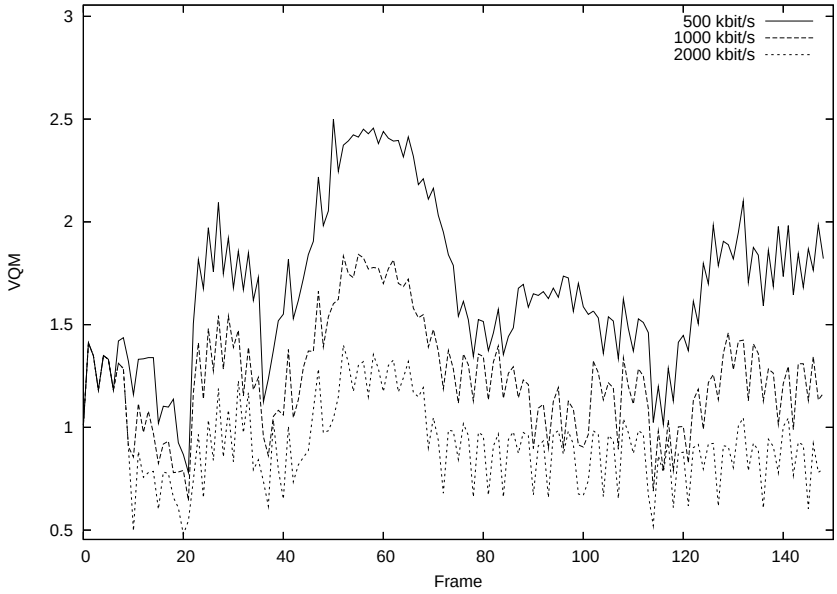

Figure B.49: VQM, daggers, 640x272, 15 fps

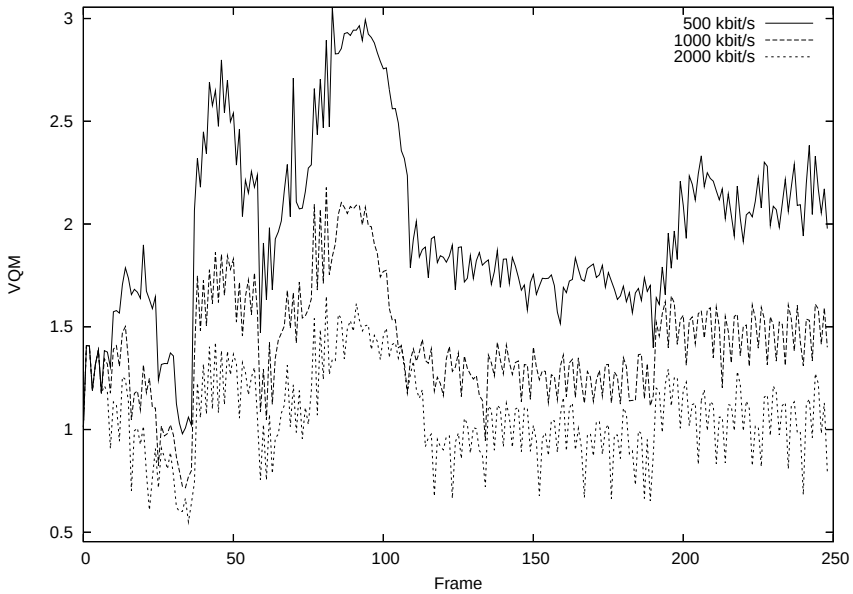

Figure B.50: VQM, daggers, 640x272, 25 fps

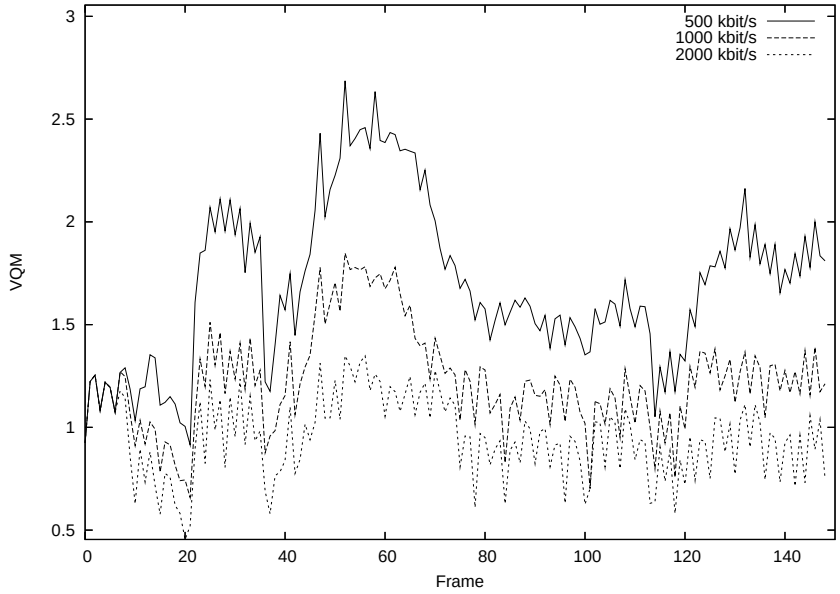

Figure B.51: VQM, daggers, 784x336, 15 fps

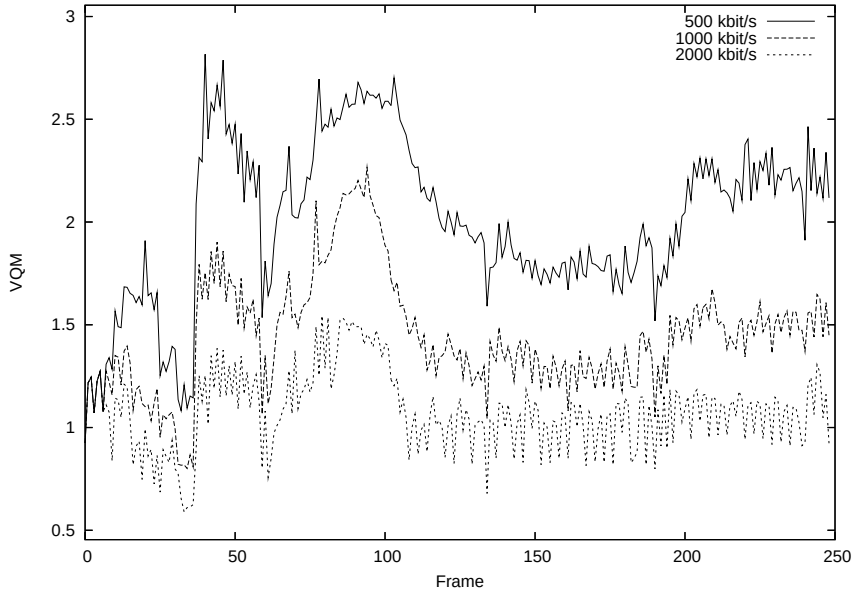

Figure B.52: VQM, daggers, 784x336, 25 fps

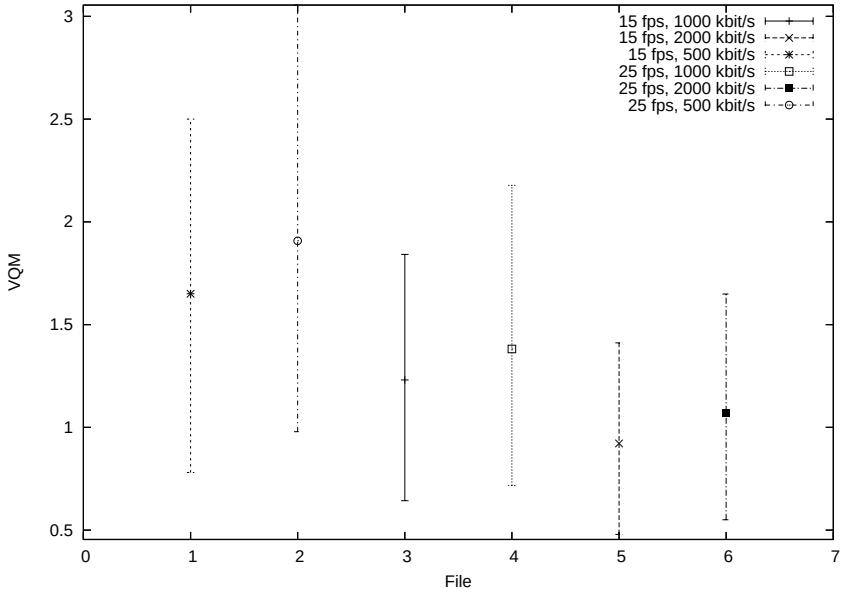

Figure B.53: VQM, daggers, 640x272, min/max/median

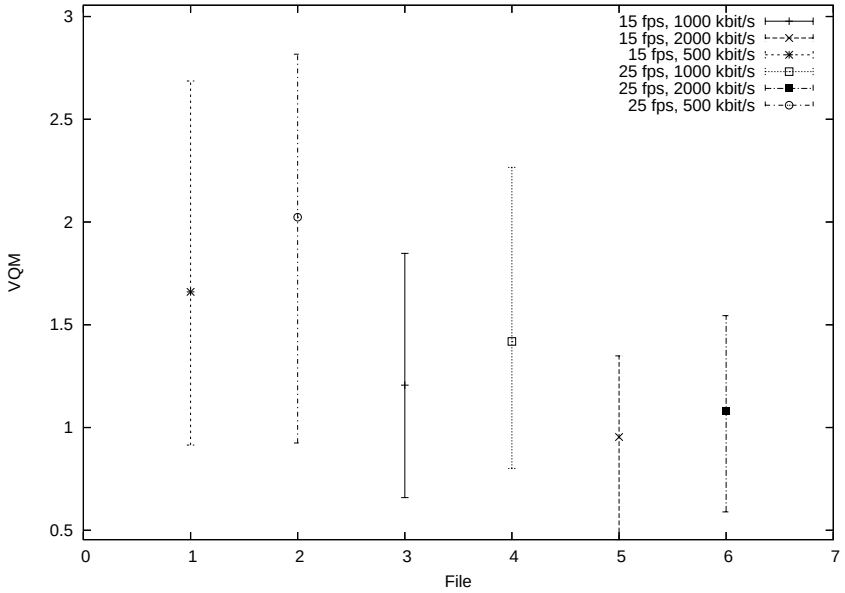

Figure B.54: VQM, daggers, 784x336, min/max/median

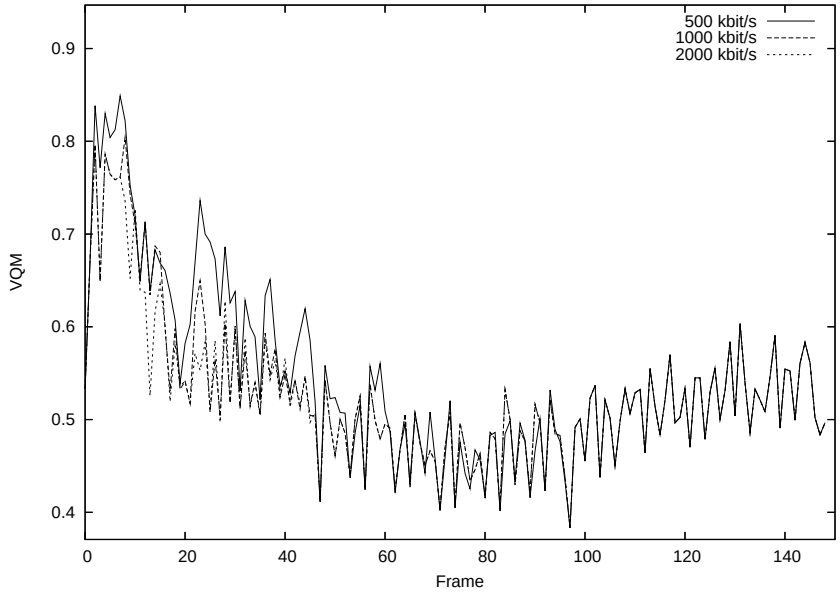

Figure B.55: VQM, evil, 592x320, 15 fps

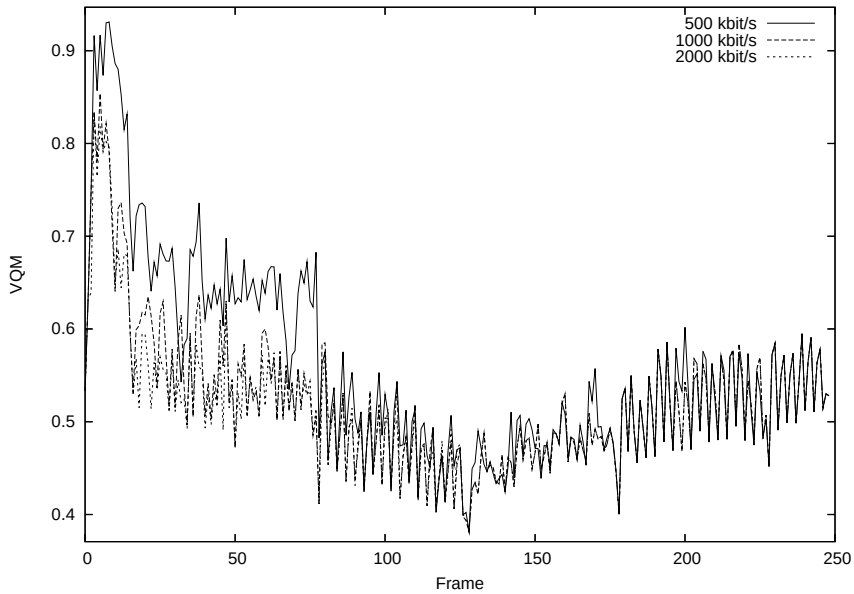

Figure B.56: VQM, evil, 592x320, 25 fps

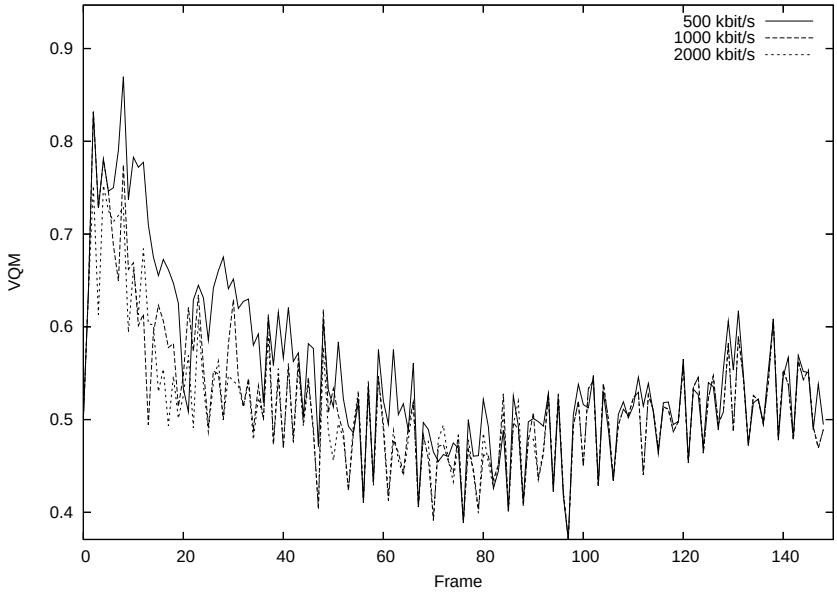

Figure B.57: VQM, evil, 704x384, 15 fps

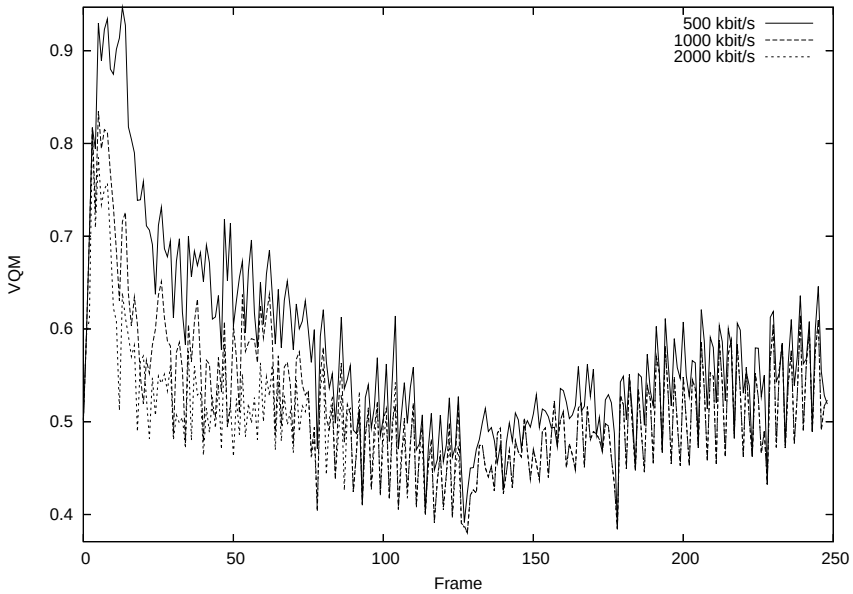

Figure B.58: VQM, evil, 704x384, 25 fps

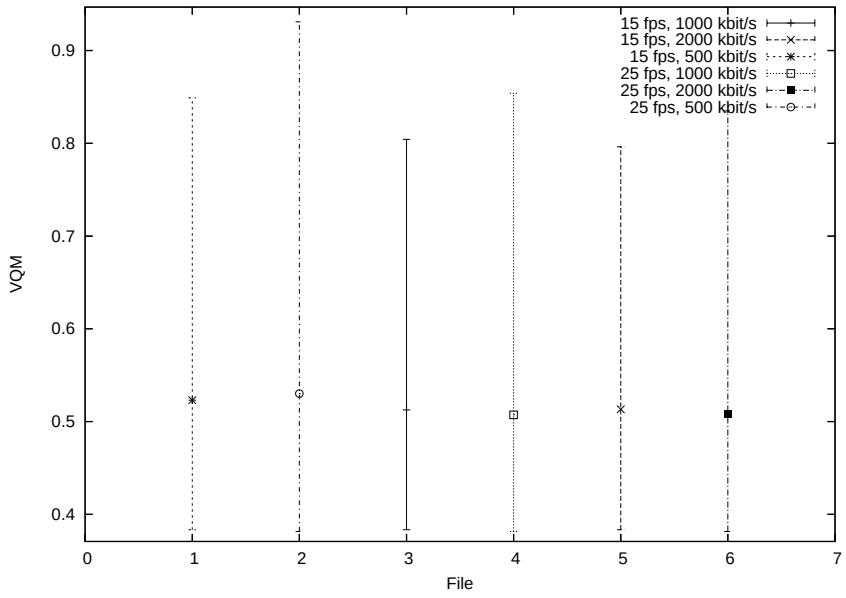

Figure B.59: VQM, evil, 592x320, min/max/median

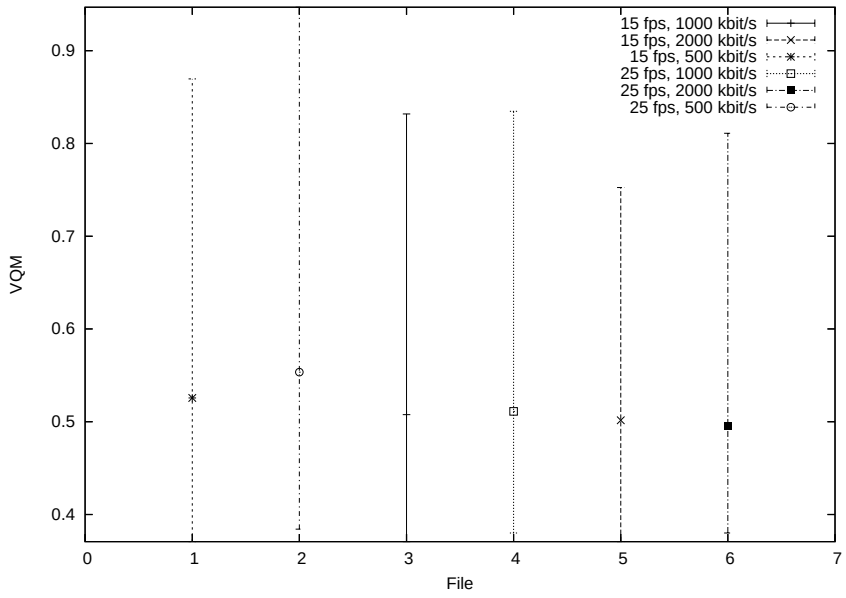

Figure B.60: VQM, evil, 704x384, min/max/median

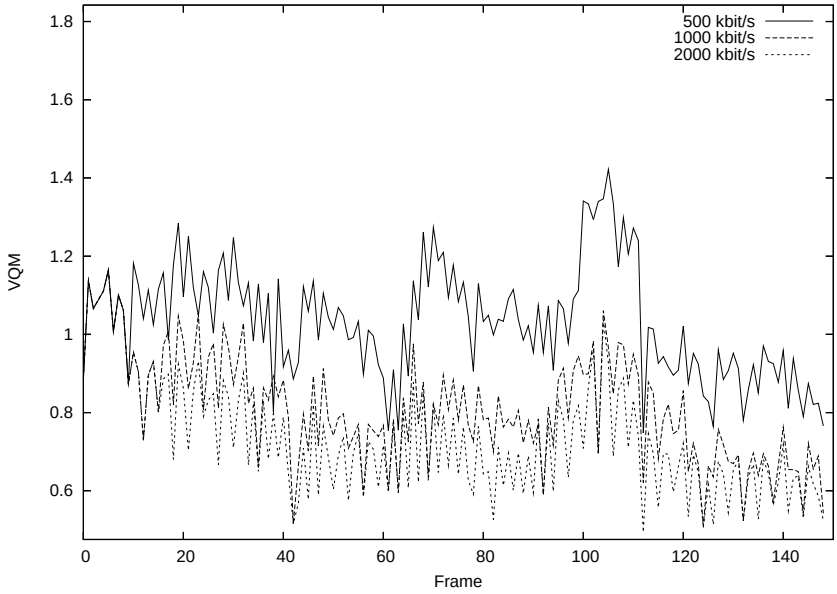

Figure B.61: VQM, gladiator, 640x272, 15 fps

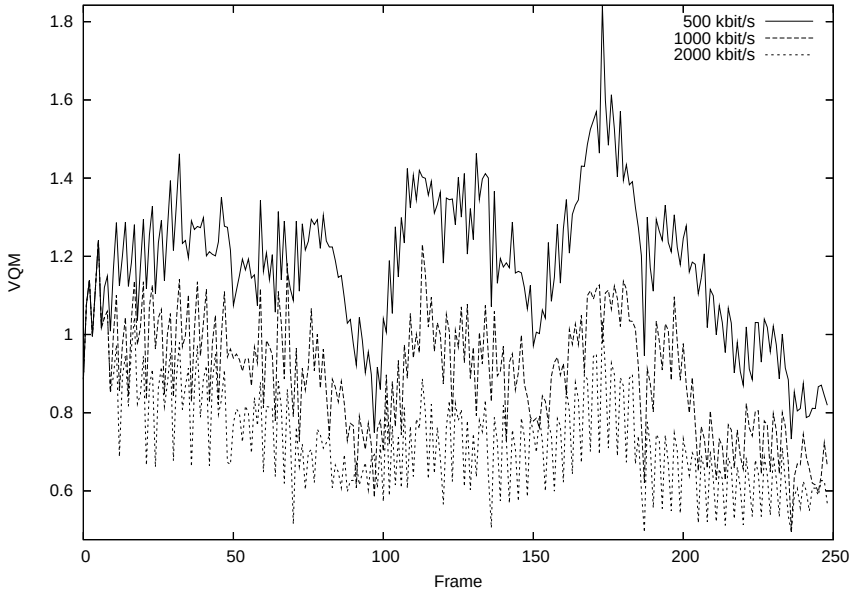

Figure B.62: VQM, gladiator, 640x272, 25 fps

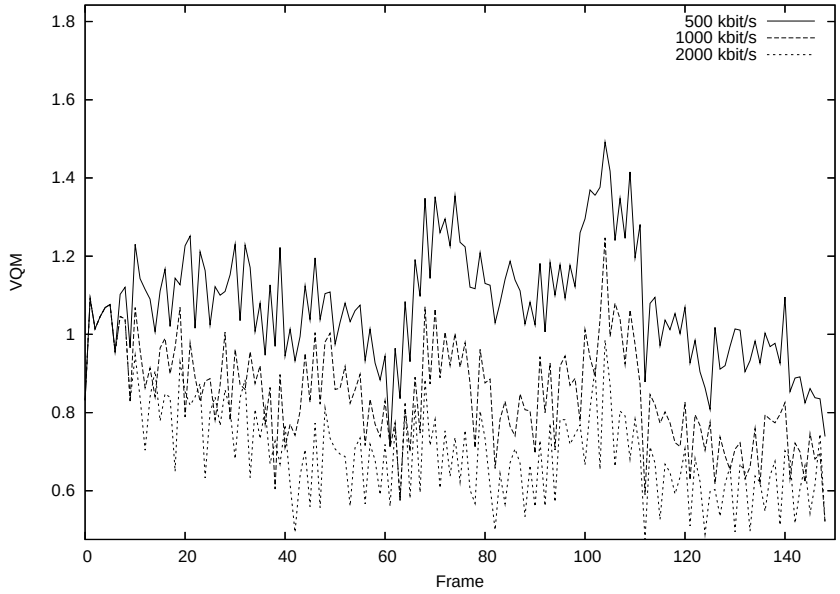

Figure B.63: VQM, gladiator, 784x336, 15 fps

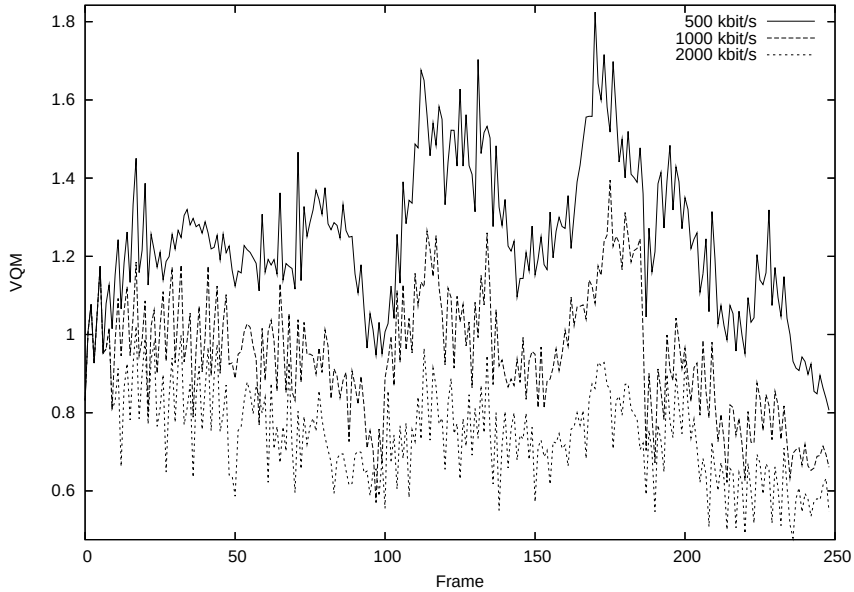

Figure B.64: VQM, gladiator, 784x336, 25 fps

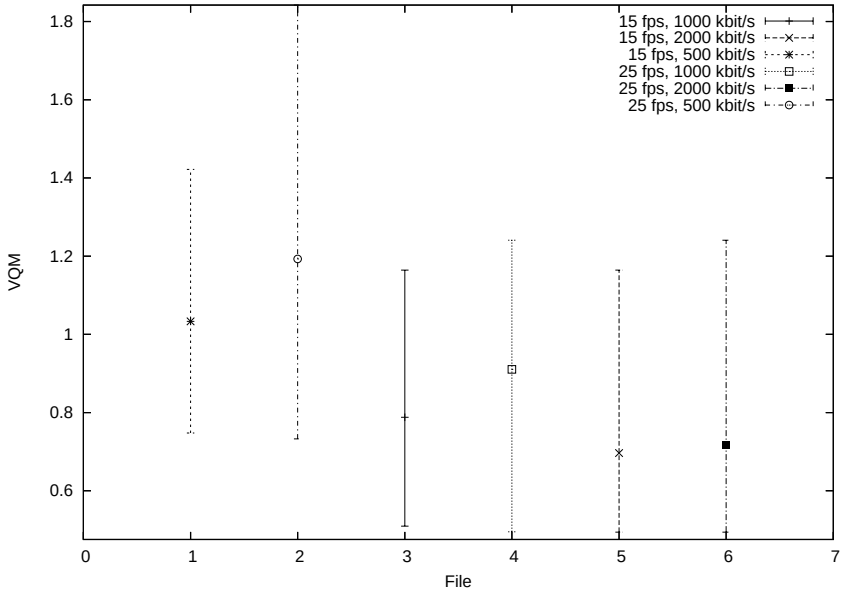

Figure B.65: VQM, gladiator, 640x272, min/max/median

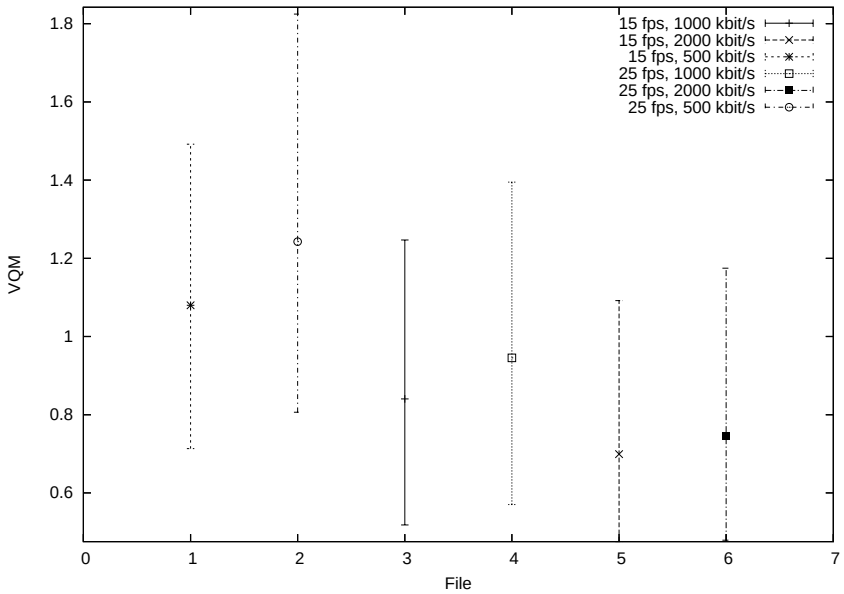

Figure B.66: VQM, gladiator, 784x336, min/max/median

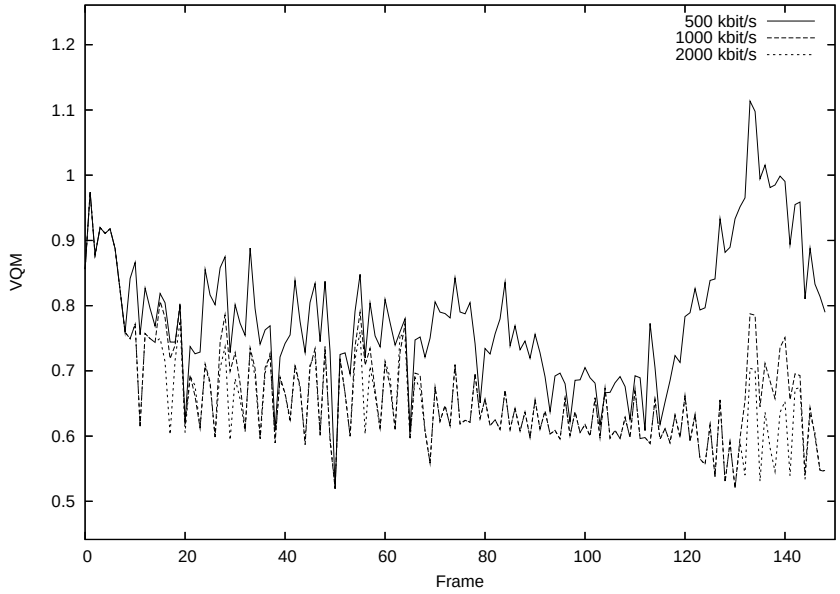

Figure B.67: VQM, princess, 592x320, 15 fps

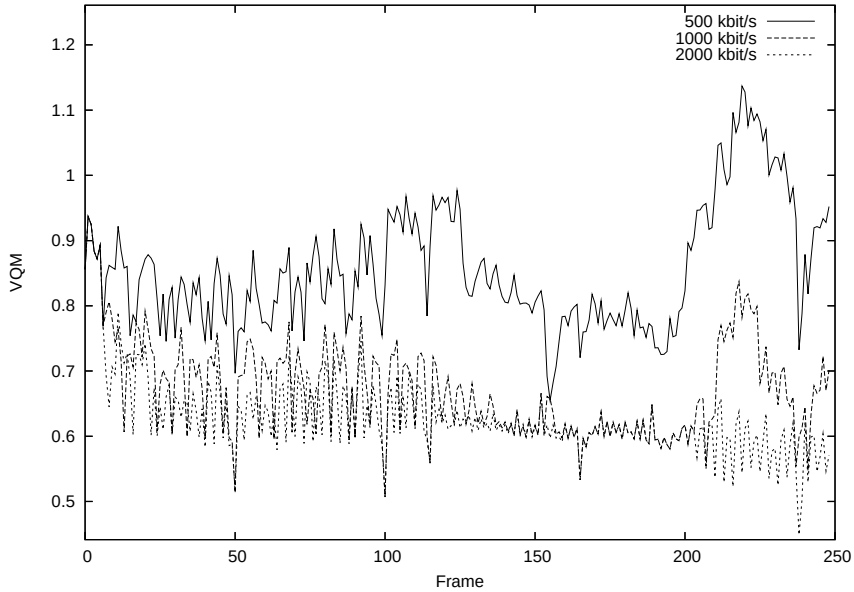

Figure B.68: VQM, princess, 592x320, 25 fps

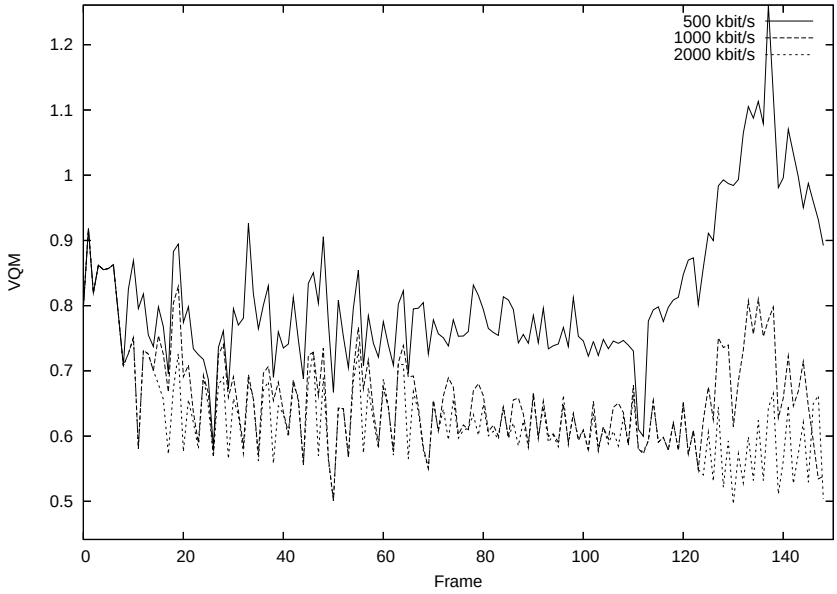

Figure B.69: VQM, princess, 704x384, 15 fps

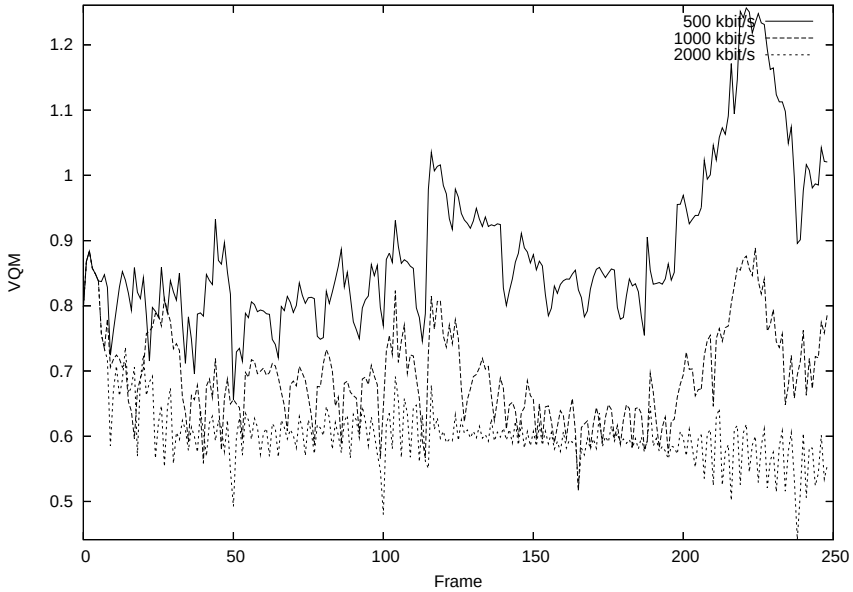

Figure B.70: VQM, princess, 704x384, 25 fps

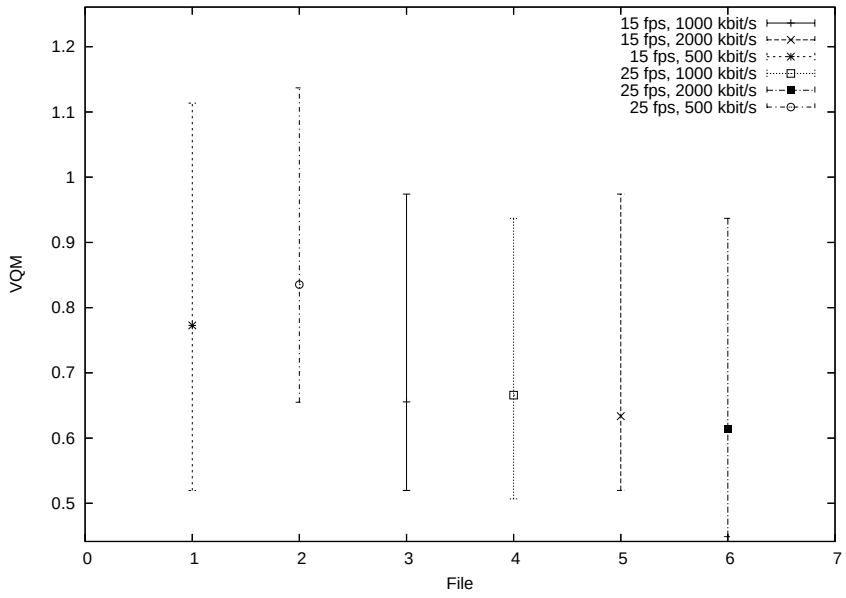

Figure B.71: VQM, princess, 592x320, min/max/median

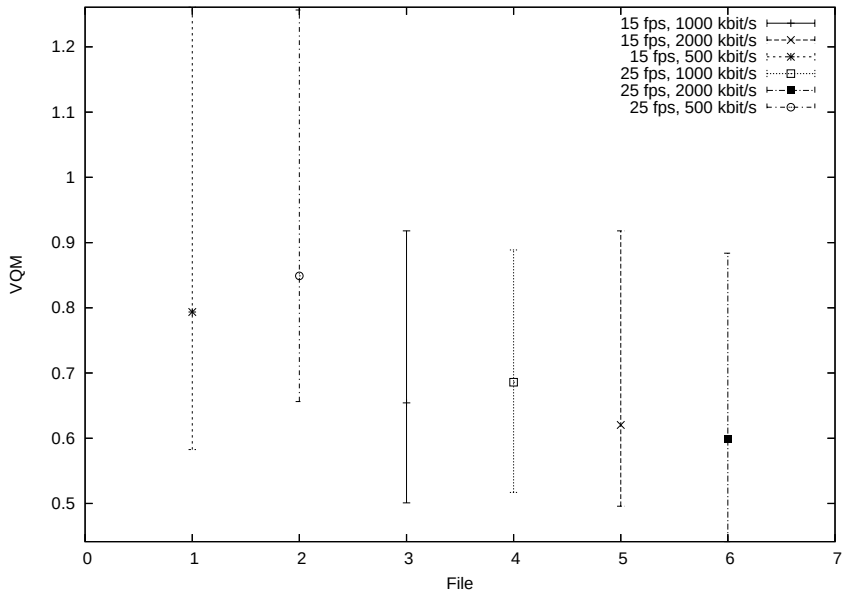

Figure B.72: VQM, princess, 704x384, min/max/median

Appendix C

Utilities

C.1 makedat.py

```
#!/bin/env python

import math
import os
import re


def plot(test, name, xmin, xmax, ymin, ymax):
    f = file("plot" + os.sep + test + "_" + name + ".plot",
        "w")
    f.write("set xlabel \"Frame\"\n")
    f.write("set ylabel \"" + test.upper() + "\"\n")
    f.write("set xrange [" + str(xmin) + ":" + str(xmax) +
       "]\n")
    f.write("set yrange [" + str(ymin) + ":" + str(ymax) +
       "]\n")
    f.write("set data style lines\n")
    terminals = {"png": ".png", "postscript eps": ".ps"}
    for term in terminals.keys():
        f.write("set output \"" + test + "_" + name +
           terminals[term] + "\"\n")
        f.write("set terminal " + term + "\n")
        f.write("plot \"dat/" + test + "_" + name + """
           _br500.dat" title '500 kbit/s', \
"dat/""" + test + "_" + name + """_br1000.dat" title '1000
    kbit/s', \
"dat/""" + test + "_" + name + "_br2000.dat\" title '2000
    kbit/s'\n")
    f.close()
```

```
def plot_avg(test, name, xmin, xmax, ymin, ymax, files):
    f = file("plot" + os.sep + name + "_avg.plot", "w")
    f.write("set xlabel \"File\"\n")
    f.write("set ylabel \"" + test.upper() + "\"\n")
    f.write("set xrange [" + str(xmin) + ":" + str(xmax) +
        "]\n")
    f.write("set yrange [" + str(ymin) + ":" + str(ymax) +
        "]\n")
    terminals = {"png": ".png", "postscript eps": ".ps"}
    for term in terminals.keys():
        f.write("set output \"" + name + "_avg" + terminals
            [term] + "\"\n")
        f.write("set terminal " + term + "\n")
        f.write("plot ")
        filecount = 1
        filetotal = len(files)
        for filename in files:
            reg = re.compile(r"(?P<test>.+)_(?P<colorspace
                >.+)_(?P<name>.+)_(?P<xsize>\d+)x(?P<ysize>\
                d+)_fps(?P<fps>\d+)_br(?P<bitrate>\d+)_avg\.
                dat")
            res = re.match(reg, filename)
            finfo = res.groupdict()

            if filecount < filetotal:
                f.write("\"dat/" + filename + "\" title '"
                    + finfo["fps"] + " fps, " + finfo["
                    bitrate"] + " kbit/s' with errorbars, ")
            else:
                f.write("\"dat/" + filename + "\" title '"
                    + finfo["fps"] + " fps, " + finfo["
                    bitrate"] + " kbit/s' with errorbars\n")
            filecount = filecount + 1
    f.close()


def plot_med(test, name, xmin, xmax, ymin, ymax, files):
    f = file("plot" + os.sep + name + "_med.plot", "w")
    f.write("set xlabel \"File\"\n")
    f.write("set ylabel \"" + test.upper() + "\"\n")
    f.write("set xrange [" + str(xmin) + ":" + str(xmax) +
        "]\n")
    f.write("set yrange [" + str(ymin) + ":" + str(ymax) +
        "]\n")
    terminals = {"png": ".png", "postscript eps": ".ps"}
    for term in terminals.keys():
        f.write("set output \"" + name + "_med" + terminals
            [term] + "\"\n")
        f.write("set terminal " + term + "\n")
```

```
        f.write("plot ")
        filecount = 1
        filetotal = len(files)
        for filename in files:
            reg = re.compile(r"(?P<test>.+)_(?P<colorspace
                >.+)_(?P<name>.+)_(?P<xsize>\d+)x(?P<ysize>\
                d+)_fps(?P<fps>\d+)_br(?P<bitrate>\d+)_med\.
                dat")
            res = re.match(reg, filename)
            finfo = res.groupdict()

            if filecount < filetotal:
                f.write("\"dat/" + filename + "\" title '"
                    + finfo["fps"] + " fps, " + finfo["
                    bitrate"] + " kbit/s' with errorbars, ")
            else:
                f.write("\"dat/" + filename + "\" title '"
                    + finfo["fps"] + " fps, " + finfo["
                    bitrate"] + " kbit/s' with errorbars\n")
            filecount = filecount + 1
    f.close()


testlist = ["psnr", "ssim", "vqm"]
# Give each fps,bitrate tuple an ID
avgid = { (15,500): 1,
          (25,500): 2,
         (15,1000): 3,
         (25,1000): 4,
         (15,2000): 5,
         (25,2000): 6 }
globalstats = {"psnr": {"min": 0, "max": 0, "avg": 0, "
    median": 0},
               "ssim": {"min": 0, "max": 0, "avg": 0, "
                   median": 0},
               "vqm": {"min": 0, "max": 0, "avg": 0, "median
                   ": 0}}
testfilenames = {}
# movie: test: name
avgfilenames = {}
medfilenames = {}

plotpath = "plot"
if not os.access("plot", os.F_OK):
    os.mkdir("plot")
datpath = plotpath + os.sep + "dat"
if not os.access(datpath, os.F_OK):
    os.mkdir(datpath)
```

```
filelist = os.listdir(os.getcwd())
for test in testlist:
    filenumber = 0
    for infile in filelist:
        if infile.startswith(test) and infile.endswith(".
            csv"):
            # Store file info
            # psnr_yyuv_gladiator_784x336_fps15_br2000.csv
            reg = re.compile(r".+_(?P<testfilename>.+_.+_\d
                +x\d+_fps\d+)_br\d+\.csv")
            res = re.match(reg, infile)
            testfilename = res.groupdict()["testfilename"]

            reg = re.compile(r"(?P<test>.+)_(?P<colorspace
                >.+)_(?P<name>.+)_(?P<xsize>\d+)x(?P<ysize>\
                d+)_fps(?P<fps>\d+)_br(?P<bitrate>\d+)\.csv"
                )
            res = re.match(reg, infile)
            finfo = res.groupdict()

            # Store test names for creating plot files
                later
            testfilenames[testfilename] = int(finfo["fps"])
                *10
            # Averages too
            if not finfo["name"] in avgfilenames:
                avgfilenames[finfo["name"]] = {}
            if not finfo["test"] in avgfilenames[finfo["
                name"]]:
                avgfilenames[finfo["name"]][finfo["test"]]
                    = {}
            res = finfo["xsize"]+"x"+finfo["ysize"]
            if not res in avgfilenames[finfo["name"]][finfo
                ["test"]]:
                avgfilenames[finfo["name"]][finfo["test"]][
                    res] = []
            avgfilenames[finfo["name"]][finfo["test"]][res
                ].append(infile[:-4] + "_avg.dat")
            # And median
            if not finfo["name"] in medfilenames:
                medfilenames[finfo["name"]] = {}
            if not finfo["test"] in medfilenames[finfo["
                name"]]:
                medfilenames[finfo["name"]][finfo["test"]]
                    = {}
            res = finfo["xsize"]+"x"+finfo["ysize"]
            if not res in medfilenames[finfo["name"]][finfo
                ["test"]]:
                medfilenames[finfo["name"]][finfo["test"]][
```

```
        res] = []
medfilenames[finfo["name"]][finfo["test"]][res
    ].append(infile[:-4] + "_med.dat")

csv = file(infile, "r")
dat = file(datpath + os.sep + infile[:-4] + ".
    dat", "w")
avg = file(datpath + os.sep + infile[:-4] + "
    _avg.dat", "w")
med = file(datpath + os.sep + infile[:-4] + "
    _med.dat", "w")

# Test type
dat.write("# " + csv.readline())
# File
line = csv.readline()
pathelements = line.split("\\")
filename = pathelements[len(pathelements)-1].
    strip()
dat.write("# " + filename + "\n")
# Rest of file
i = 0  # Framecount
min = 0
max = 0
values = []
for line in csv.readlines():
    if line[:3] == "AVG":
        # Find delta between max/min
        delta = max - abs(min)
        text, value = line.split()
        avg.write("# " + test + " average\n")
        avg.write("# " + filename + "\n")
        avg.write(str(avgid[(int(finfo["fps"]),
            int(finfo["bitrate"]))]) + "\t" +
            value + "\t" + str(min) + "\t" + str
            (max) + "\n")
        avg.close()
    else:
        num = float(line)
        # Store value for median calc
        values.append(num)
        # Find min/max
        if i == 0:
            min = num
            max = num
        else:
            if num > max:
                max = num
            elif num < min:
```

```
                        min = num
                    # Update global statistics
                    if filenumber == 0:
                        globalstats[test]["min"] = num
                        globalstats[test]["max"] = num
                    else:
                        if num < globalstats[test]["min"]:
                            globalstats[test]["min"] = num
                        elif num > globalstats[test]["max"
                            ]:
                            globalstats[test]["max"] = num

                    dat.write(str(i) + "\t" + line)
                    i = i + 1
                # Find and write median
                values.sort()
                medpos = float(len(values)-1) / 2
                median = (values[int(math.floor(medpos))] +
                    values[int(math.ceil(medpos))]) / 2
                med.write("# " + test + " median\n")
                med.write("# " + filename + "\n")
                med.write(str(avgid[(int(finfo["fps"]),int(
                    finfo["bitrate"]))]) + "\t" + str(median) +
                    "\t" + str(min) + "\t" + str(max) + "\n")
                med.close()

                dat.close()
                csv.close()
                filenumber = filenumber + 1

        print test + " stats:"
        print "min: " + str(globalstats[test]["min"]) + ", max:
            " + str(globalstats[test]["max"])

# Create plots
for name in testfilenames.keys():
    plot("psnr", name, 0, testfilenames[name], math.floor(
        globalstats["psnr"]["min"]), math.ceil(globalstats["
        psnr"]["max"]))
    plot("ssim", name, 0, testfilenames[name], globalstats[
        "ssim"]["min"], 1.0)
    plot("vqm", name, 0, testfilenames[name], globalstats["
        vqm"]["min"], globalstats["vqm"]["max"])

# Average plots
for clip in avgfilenames.keys():
    for test in avgfilenames[clip].keys():
        for res in avgfilenames[clip][test].keys():
            name = test + "_yyuv_" + clip + "_" + res
```

```
            plot_avg(test, name, 0, len(avgfilenames[clip][
                test][res])+1, globalstats[test]["min"],
                globalstats[test]["max"], avgfilenames[clip
                ][test][res])

# Median plots
for clip in medfilenames.keys():
    for test in medfilenames[clip].keys():
        for res in medfilenames[clip][test].keys():
            name = test + "_yyuv_" + clip + "_" + res
            plot_med(test, name, 0, len(medfilenames[clip][
                test][res])+1, globalstats[test]["min"],
                globalstats[test]["max"], medfilenames[clip
                ][test][res])
```

C.2 makedscqsdat.py

```
#!/bin/env python

import math
import os
import re
import sys


# Parse resultfile
if os.access(sys.argv[1], os.F_OK):
    resultfile = file(sys.argv[1], "r")
else:
    print "No such file!"
    sys.exit()


# Contains scores, watched count, min, max
filestats = {}

# Classify resolutions
resclass = {"640x272": "medium",
            "592x320": "medium",
            "704x384": "high",
            "784x336": "high"}

# Give each fps,bitrate tuple an ID
videoid = { (15,500): 1,
           (15,1000): 2,
           (15,2000): 3,
            (25,500): 4,
           (25,1000): 5,
           (25,2000): 6 }
```

```
def plot_med(path, name, xmin, xmax, ymin, ymax, files):
    f = file("plot" + os.sep + name + "_med.plot", "w")
    f.write("set xlabel \"File\"\n")
    f.write("set ylabel \"Score\"\n")
    f.write("set xrange [" + str(xmin) + ":" + str(xmax) +
        "]\n")
    f.write("set yrange [" + str(ymin) + ":" + str(ymax) +
        "]\n")
    terminals = {"png": ".png", "postscript eps": ".ps"}
    for term in terminals.keys():
        f.write("set output \"" + name + "_med" + terminals
            [term] + "\"\n")
        f.write("set terminal " + term + "\n")
        f.write("plot ")
        filecount = 1
        filetotal = len(files)
        for filename in files:
            reg = re.compile(r"(?P<class>.+)_fps(?P<fps>\d
                +)_br(?P<bitrate>\d+)")
            res = re.match(reg, filename)
            finfo = res.groupdict()

            if filecount < filetotal:
                f.write("\"dat/" + filename + "_med.dat\"
                    title '" + finfo["fps"] + " fps, " +
                    finfo["bitrate"] + " kbit/s' with
                    errorbars, ")
            else:
                f.write("\"dat/" + filename + "_med.dat\"
                    title '" + finfo["fps"] + " fps, " +
                    finfo["bitrate"] + " kbit/s' with
                    errorbars\n")
            filecount = filecount + 1
    f.close()


# Resultfile structure:
# princess_704x384_fps25_br2000.mp4,reference
# 4,5
# 3

line = resultfile.readline()
while line:
    # Get tested file name
    fileA, fileB = line.strip().split(",")
    if fileA == "reference":
        tested = fileB
```

```
else:
    tested = fileA
# Get info about tested file
reg = re.compile(r"(?P<name>.+)_(?P<xsize>\d+)x(?P<
    ysize>\d+)_fps(?P<fps>\d+)_br(?P<bitrate>\d+).mp4")
res = re.match(reg, tested)
finfo = res.groupdict()

# Get score
line = resultfile.readline()
scoreA, scoreB = line.strip().split(",")
score = abs(int(scoreA) - int(scoreB))

# Remove name and absolute resolution from filename
resolution = finfo["xsize"] + "x" + finfo["ysize"]
resname = resclass[resolution]
key = tested.replace(finfo["name"] + "_", "", 1)
key = key.replace(resolution, resname, 1)
key = key[:-4]  # Remove .mp4 suffix
if not key in filestats:
    filestats[key] = {}
if "scores" in filestats[key]:
    filestats[key]["scores"].append(score)
else:
    filestats[key]["scores"] = [score]

if not "score_min" in filestats[key]:
    filestats[key]["score_min"] = score
    filestats[key]["score_max"] = score
elif score < filestats[key]["score_min"]:
    filestats[key]["score_min"] = score
elif score > filestats[key]["score_max"]:
    filestats[key]["score_max"] = score

# Get times watched
line = resultfile.readline()
watched = int(line.strip())
if "watched" in filestats[key]:
    filestats[key]["watched"].append(watched)
else:
    filestats[key]["watched"] = [watched]

if not "watched_min" in filestats[key]:
    filestats[key]["watched_min"] = watched
    filestats[key]["watched_max"] = watched
elif watched < filestats[key]["watched_min"]:
    filestats[key]["watched_min"] = watched
elif watched > filestats[key]["watched_max"]:
    filestats[key]["watched_max"] = watched
```

```
    line = resultfile.readline()


resultfile.close()
print str(filestats)

plotdir = "plot"
datdir = plotdir + os.sep + "dat"
if not os.access(plotdir, os.F_OK):
    os.mkdir(plotdir)
if not os.access(datdir, os.F_OK):
    os.mkdir(datdir)

# Derive statistics
for name in filestats.keys():
    # Get info from name
    reg = re.compile(r"(?P<class>.+)_fps(?P<fps>\d+)_br(?P<
        bitrate>\d+)")
    res = re.match(reg, name)
    finfo = res.groupdict()
    id = videoid[(int(finfo["fps"]),int(finfo["bitrate"]))]

    # Scores data file
    datfile = file(datdir + os.sep + name + ".dat", "w")
    datfile.write("# DSCQS scores\n")
    datfile.write("# " + name + "\n")
    i = 0
    sum = 0
    for score in filestats[name]["scores"]:
        datfile.write(str(i) + "\t" + str(filestats[name]["
            scores"][i]) + "\n")
        sum = sum + score
        i = i + 1
    datfile.close()

    # Average data file
    datfile = file(datdir + os.sep + name + "_avg.dat", "w"
        )
    datfile.write("# DSCQS average\n")
    datfile.write("# " + name + "\n")
    datfile.write(str(id) + "\t" \
                  + str(float(sum) / i) + "\t" \
                  + str(filestats[name]["score_min"]) + "\t
                      " \
                  + str(filestats[name]["score_max"]) + "\n
                      ")
    datfile.close()
```

```
    # Median data file
    values = filestats[name]["scores"]
    values.sort()
    medpos = float(len(values)-1) / 2
    median = float(values[int(math.floor(medpos))] + values
        [int(math.ceil(medpos))]) / 2
    datfile = file(datdir + os.sep + name + "_med.dat", "w"
        )
    datfile.write("# DSCQS median\n")
    datfile.write("# " + name + "\n")
    datfile.write(str(id) + "\t" \
                  + str(median) + "\t" \
                  + str(filestats[name]["score_min"]) + "\t
                      " \
                  + str(filestats[name]["score_max"]) + "\n
                      ")
    datfile.close()

# Create plots
for resclass in ["medium", "high"]:
    files = []
    for name in filestats.keys():
        if name.startswith(resclass):
            files.append(name)
    files.sort()
    plot_med(plotdir, resclass, 0, len(files)+1, 0, 5,
        files)
```

C.3 massencode.py

```
#!/bin/env python

import os
import sys


if not len(sys.argv) == 2:
    print "Usage: " + sys.argv[0] + " aspect"
    sys.exit()

aspect = sys.argv[1]

bitrates = {"low": 500, "mid": 1000, "high": 2000}

mencoderparam = "\"%(avs)s\" -ovc xvid -xvidencopts bitrate
    =%(bitrate)s:max_key_interval=50:vhq=4:chroma_me:trellis
    :lumi_mask:bvhq=1:rc_reaction_delay_factor=16:
    rc_averaging_period=100:chroma_opt:hq_ac -o \"%(m4v)s\"
    -of rawvideo"
```

```
cwd = os.getcwd()
encodepath = os.path.join(cwd, "encodes")
files = os.listdir(cwd)

for file in files:
    if file[-4:] == ".avs":
        # Get attribs: name, size, fps
        name, size, fps = file[:-4].split("_")
        # Check if target path for encodes exists
        if not os.access(encodepath, os.F_OK):
            print "Creating " + encodepath
            os.mkdir(encodepath)

        for br in bitrates.values():
            basename = name + "_" + size + "_" + "fps" +
                fps + "_" + "br" + str(br)
            m4v = os.path.join(encodepath, basename + ".m4v
                ")
            mp4 = os.path.join(encodepath, basename + ".mp4
                ")
            avs = os.path.join(encodepath, basename + ".avs
                ")

            if not os.access(mp4, os.F_OK):
                videocmd = "cmd /C C:\\bin\\mencoder.exe "
                    + \
                    mencoderparam % \
                        {"avs": file, "bitrate": br, "m4v":
                            m4v}
                #print videocmd
                ret = os.system(videocmd)

                # Mux video to mp4
                muxcmd = "cmd /C C:\\bin\\MP4Box.exe " + \
                    "-nodrop -fps " + fps + " -add \"" +
                        m4v + "\" \"" + mp4 + "\""
                #print muxcmd
                ret = os.system(muxcmd)

                # Delete m4v file
                os.remove(m4v)
            else:
                print "Exists: " + mp4
            # Write AVS file which encapsulates the MP4
                file
            if not os.access(avs, os.F_OK):
                avsfile = open(avs, "w")
                avsfile.write("DirectShowSource(\"" + mp4 +
```

```
                "\", fps=" + fps + ")\n")
            avsfile.close()
        else:
            print "Exists: " + avs
```

C.4 subjective.py

```
#!/bin/env python

import glob
import os
import random
import re

import wx
import cElementTree as ElementTree


class NameDialog(wx.Dialog):
    """Asks for name/ID input."""

    def __init__(
            self, parent, ID, title, size=wx.DefaultSize,
                pos=wx.DefaultPosition,
            style=wx.DEFAULT_DIALOG_STYLE
            ):
        wx.Dialog.__init__(self, parent, ID, title, pos,
            size, style)

        brd = 5  # Border size
        s = wx.BoxSizer(wx.HORIZONTAL)

        label = wx.StaticText(self, -1, u"Name or ID:")
        s.Add(label, flag=wx.ALL|wx.ALIGN_CENTER, border=
            brd)

        self.userid = wx.TextCtrl(self, -1, u"", size=(125,
             -1))
        s.Add(self.userid, flag=wx.ALL|wx.ALIGN_CENTER,
            border=brd)

        button = wx.Button(self, wx.ID_OK, u"Enter")
        s.Add(button, flag=wx.ALL|wx.ALIGN_CENTER, border=
            brd)

        self.SetSizer(s)
        self.Fit()
```

```
class ScoreDialog(wx.Dialog):
    """Asks for comparison score."""

    def __init__(
            self, parent, ID, title, size=wx.DefaultSize,
                pos=wx.DefaultPosition,
            style=wx.DEFAULT_DIALOG_STYLE
            ):
        wx.Dialog.__init__(self, parent, ID, title, pos,
            size, style)

        brd = 5
        sizer = wx.BoxSizer(wx.VERTICAL)
        s = wx.BoxSizer(wx.HORIZONTAL)

        vsizer = wx.BoxSizer(wx.VERTICAL)
        vsizer.Add(wx.StaticText(self, -1, u"A"), flag=wx.
            ALL|wx.ALIGN_CENTER, border=brd)
        self.sliderA = wx.Slider(
            self, -1, 3, 1, 5, size=(-1, 200),
            style=wx.SL_RIGHT | wx.SL_INVERSE | wx.
                SL_AUTOTICKS | wx.SL_LABELS
            )
        vsizer.Add(self.sliderA, flag=wx.ALL|wx.
            ALIGN_CENTER, border=brd)
        s.Add(vsizer, flag=wx.EXPAND|wx.ALIGN_CENTER)

        vsizer = wx.BoxSizer(wx.VERTICAL)
        vsizer.Add(wx.StaticText(self, -1, u"B"), flag=wx.
            ALL|wx.ALIGN_CENTER, border=brd)
        self.sliderB = wx.Slider(
            self, -1, 3, 1, 5, size=(-1, 200),
            style=wx.SL_LEFT | wx.SL_INVERSE | wx.
                SL_AUTOTICKS | wx.SL_LABELS
            )
        vsizer.Add(self.sliderB, flag=wx.ALL|wx.
            ALIGN_CENTER, border=brd)
        s.Add(vsizer, flag=wx.EXPAND|wx.ALIGN_CENTER)

        sizer.Add(s, flag=wx.ALIGN_CENTER)
        sizer.Add(wx.Button(self, wx.ID_OK, label=u"Done"),
            flag=wx.ALL|wx.ALIGN_CENTER, border=brd)

        vsizer = wx.BoxSizer(wx.VERTICAL)
        vsizer.Add(wx.StaticText(self, -1, u"Legend:"),
            flag=wx.ALL|wx.ALIGN_LEFT|wx.
            ALIGN_CENTER_VERTICAL, border=brd)
        vsizer.Add(wx.StaticText(self, -1, u"5 - Perfect"),
```

```
            flag=wx.ALL|wx.ALIGN_LEFT|wx.
            ALIGN_CENTER_VERTICAL, border=brd)
        vsizer.Add(wx.StaticText(self, -1, u"4 - Good"),
            flag=wx.ALL|wx.ALIGN_LEFT|wx.
            ALIGN_CENTER_VERTICAL, border=brd)
        vsizer.Add(wx.StaticText(self, -1, u"3 - Average"),
            flag=wx.ALL|wx.ALIGN_LEFT|wx.
            ALIGN_CENTER_VERTICAL, border=brd)
        vsizer.Add(wx.StaticText(self, -1, u"2 - Bad"),
            flag=wx.ALL|wx.ALIGN_LEFT|wx.
            ALIGN_CENTER_VERTICAL, border=brd)
        vsizer.Add(wx.StaticText(self, -1, u"1 - Horrible")
            , flag=wx.ALL|wx.ALIGN_LEFT|wx.
            ALIGN_CENTER_VERTICAL, border=brd)
        sizer.Add(vsizer, flag=wx.ALIGN_CENTER)

        self.SetSizer(sizer)


class SubjectiveTestFrame(wx.Frame):
    """SubjectiveTest main wx.Frame."""

    def __init__(self, parent, ID, title):
        wx.Frame.__init__(self, parent, ID, title, size
            =(250,225), pos=wx.DefaultPosition)
        # Config
        config = ElementTree.parse("subjective.xml")
        clips = ElementTree.parse("clips.xml")
        self.clippath = config.findtext("clippath")
        self.resultpath = config.findtext("resultpath")
        self.mplayer = config.findtext("mplayer")
        # Encodepath is clippath+clipname+encodepath
        self.encodepath = "encodes"
        # Directories
        self.maindir = ""
        self.logdir = ""
        # Globals
        self.clipnameA = ""
        self.clipnameB = ""
        self.watchcount = 0
        self.userid = u"dscqs"
        self.clipnames = []
        self.clipnumber = 0
        self.encodelist = []
        self.avs = ""
        # GUI Globals
        self.usertext = None
        self.testtext = None
```

```
        self.watchbutton = None
        self.donebutton = None
        # Main GUI
        panel = wx.Panel(self, -1)
        sizer = wx.BoxSizer(wx.VERTICAL)
        brd = 5

        self.usertext = wx.StaticText(panel, -1, u"
            Participant: ")
        sizer.Add(self.usertext, flag=wx.ALL|wx.
            ALIGN_CENTER, border=brd)

        self.testtext = wx.StaticText(panel, -1, u"Test: ")
        sizer.Add(self.testtext, flag=wx.ALL|wx.
            ALIGN_CENTER, border=brd)

        info = u"""Welcome to the subjective DSCQS test.
To view the video pairs press the 'Watch'
button. On the first viewing they will
be shown twice. Press 'Done' when you
want to evaluate them."""
        sizer.Add(wx.StaticText(panel, -1, info), flag=wx.
            ALL|wx.ALIGN_CENTER, border=brd)

        self.watchbutton = wx.Button(panel, -1, size
            =(100,-1), label=u"Watch (" + str(self.
            watchcount) + " times)")
        panel.Bind(wx.EVT_BUTTON, self.OnWatchButton, self.
            watchbutton)
        sizer.Add(self.watchbutton, flag=wx.ALL|wx.
            ALIGN_CENTER, border=brd)

        self.donebutton = wx.Button(panel, -1, size
            =(100,-1), label=u"Done")
        self.donebutton.Disable()
        panel.Bind(wx.EVT_BUTTON, self.OnDoneButton, self.
            donebutton)
        sizer.Add(self.donebutton, flag=wx.ALL|wx.
            ALIGN_CENTER, border=brd)

        panel.SetSizer(sizer)

        # Get all clip names, and prepare for first test
        cliplist = clips.findall("clip")
        for c in cliplist:
            self.clipnames.append(c.get("name"))
        # Pick a random clip to start with
        self.clipnumber = random.randint(0, len(self.
            clipnames)-1)
```

```
        name = self.clipnames[self.clipnumber]
        path = self.clippath + os.sep + name + os.sep +
            self.encodepath
        # Pick random encoding for that clip
        self.encodelist = glob.glob(path + os.sep + "*.mp4"
            )
        self.encodenumber = random.randint(0, len(self.
            encodelist)-1)
        self.createAVS(name + "_reference.avs", self.
            encodelist[self.encodenumber])
        # Update GUI
        self.testtext.SetLabel(self.testtext.GetLabel() +
            name)
        self.Centre()


    def OnWatchButton(self, evt):
        clipfile = self.clippath + os.sep + self.clipnames[
            self.clipnumber] + os.sep + self.clipnames[self.
            clipnumber] + "_test.avs"
        if self.watchcount == 0:
            while self.watchcount < 2:
                os.system(self.mplayer + " " + clipfile)
                self.watchcount = self.watchcount + 1
            self.donebutton.Enable()
        else:
            os.system(self.mplayer + " " + clipfile)
            self.watchcount = self.watchcount + 1
        self.watchbutton.SetLabel(u"Watch (" + str(self.
            watchcount) + " times)")


    def OnDoneButton(self, evt):
        dlg = ScoreDialog(self, -1, u"Evaluate clips", size
            =(175,450))
        ret = dlg.ShowModal()
        # If window was closed, do nothing
        if ret == wx.ID_OK:
            name = self.clipnames[self.clipnumber]
            # Store score
            resfile = file(self.resultpath + os.sep + self.
                userid + "_results.txt", "a")
            resfile.write(self.clipnameA + "," + self.
                clipnameB + "\n")
            resfile.write(str(dlg.sliderA.GetValue()) + ","
                + str(dlg.sliderB.GetValue()) + "\n")
            resfile.write(str(self.watchcount) + "\n")
            resfile.close()
```

```
            # Go to next test
            os.remove(self.clippath + os.sep + name + os.
                sep + name + "_test.avs")
            del self.clipnames[self.clipnumber]
            if len(self.clipnames) == 0:
                self.Close()
                return
            self.clipnumber = random.randint(0, len(self.
                clipnames)-1)
            name = self.clipnames[self.clipnumber]
            path = self.clippath + os.sep + name + os.sep +
                 self.encodepath
            # Pick random encoding for that clip
            self.encodelist = glob.glob(path + os.sep + "*.
                mp4")
            self.encodenumber = random.randint(0, len(self.
                encodelist)-1)
            self.createAVS(name + "_reference.avs", self.
                encodelist[self.encodenumber])
            # Update GUI
            self.testtext.SetLabel("Test: " + name)
            self.watchcount = 0
            self.watchbutton.SetLabel(u"Watch (" + str(self
                .watchcount) + " times)")
            self.donebutton.Disable()
        dlg.Destroy()


    def createAVS(self, reference, clip):
        """Creates the comparison AVS script.

        Parameters:
        reference - Name of reference file (with full path)
            .
        clip - Name of comparison clip (with full path).
        """

        clipinfo = ClipInfo(os.path.basename(clip))
        self.avs = self.clippath + os.sep + clipinfo.name +
             os.sep + clipinfo.name + "_test.avs"
        avstext = "blank = BlankClip(width=" + clipinfo.
            xsize + ", height=" + clipinfo.ysize + ", length
            =100, fps=25.0, pixel_type=\"YV12\")\n"
        avstext = avstext + "blank = KillAudio(blank)\n"

        refclip = "Import(\"" + self.clippath + os.sep \
                              + clipinfo.name + os.sep \
                              + clipinfo.name + "_reference
                                  .avs" + "\")\n"
```

```
        otherclip = "other = DirectShowSource(\"" + clip +
            "\", fps=" + clipinfo.fps + ")\n"
        # Choose which goes first, ref or other
        cliporder = [None, None]
        rnd = random.randint(0,1)
        if rnd == 0:
            cliporder[0] = refclip + "clipA = reference\n"
            self.clipnameA = "reference"
            cliporder[1] = otherclip + "clipB = other\n"
            self.clipnameB = clipinfo.filename
        else:
            cliporder[0] = otherclip + "clipA = other\n"
            self.clipnameA = clipinfo.filename
            cliporder[1] = refclip + "clipB = reference\n"
            self.clipnameB = "reference"

        idlist = ["A", "B"]
        for i in range(2):
            blank = "blank" + idlist[i] + " = Subtitle(
                blank, \"Clip " + idlist[i] + "\", size=32,
                align=5)\n"
            avstext = avstext + blank + cliporder[i] \
                      + "clip" + idlist[i] + " =
                          LanczosResize(clip" + idlist[i] +
                          ", " + clipinfo.xsize + "," +
                          clipinfo.ysize + ")\n" \
                      + "clip" + idlist[i] + " = Subtitle(
                          clip" + idlist[i] + ", \"Clip " +
                          idlist[i] + "\")\n" \
                      + "clip" + idlist[i] + " = ChangeFPS(
                          clip" + idlist[i] + ", 25.0)\n"

        avstext = avstext + "final = blankA + clipA +
            blankB + clipB\n" \
                  + "return final\n"
        avsfile = file(self.avs, "w")
        avsfile.write(avstext)
        avsfile.close()



class ClipInfo:
    """Extracts info from a clip name."""

    def __init__(self, clip):
        reg = re.compile(r"(?P<name>.+)_(?P<xsize>\d+)x(?P<
            ysize>\d+)_fps(?P<fps>\d+)_br(?P<bitrate>\d+)")
        res = re.match(reg, clip)
        finfo = res.groupdict()
```

```
        self.name = finfo["name"]
        self.xsize = finfo["xsize"]
        self.ysize = finfo["ysize"]
        self.fps = finfo["fps"]
        self.bitrate = finfo["bitrate"]
        self.filename = clip


class SubjectiveTest(wx.App):
    """Main SubjectiveTest application class."""

    def OnInit(self):
        self.frame = SubjectiveTestFrame(None, -1, u"
            SubjectiveTest v0.1")
        # Get name/id
        #dlg = NameDialog(self.frame, -1, u"Input name or
            ID")
        #dlg.ShowModal()
        #self.frame.userid = dlg.userid.GetValue()
        self.frame.usertext.SetLabel(self.frame.usertext.
            GetLabel() + self.frame.userid)
        #dlg.Destroy()
        self.frame.Show(True)
        self.SetTopWindow(self.frame)
        return True


if __name__ == "__main__":
    app = SubjectiveTest(0)
    app.MainLoop()
```